CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Elsevier - Publisher Connector

J. Symbolic Computation (1998) **25**, 759–793
Article No. sy970198

Infinite String Rewrite Systems and Complexity†

JEAN-CAMILLE BIRGET‡

Department of Computer Science, University of Nebraska, Lincoln, NE 68588, U.S.A.

We study the relation between time complexity and derivation work for the word problem of infinitely presented semigroups and groups. We introduce the notion of the *work* of a derivation (defined as the sum of the lengths of all the rules used in the derivation, with multiplicity). The following results are proved:

(1) A finitely generated semigroup S has a decidable word problem iff S is embeddable into a finitely generated semigroup with a *complete* (i.e. confluent and terminating) presentation whose set of rewrite rules forms a finite-state sequential partial function.
A finitely generated semigroup S has a representative function which is computable in deterministic time $\leq T(.)^{O(1)}$ and has a linear upper bound on its length, iff S is *embeddable* in a semigroup with a *complete* presentation $\langle A : R\rangle$ where A is finite, R is a finite-state sequential partial function, every derivation has work $\leq T(.)^{O(1)}$, and reduction does not increase lengths more than linearly.

(2) The word problem of a finitely generated monoid (or group) S is decidable in nondeterministic time $\leq T(.)^{O(1)}$ iff S has a (group) presentation $\langle A : R\rangle$ where A is finite, R is the intersection of two deterministic context-free languages, and the minimum derivation work between equivalent words x, y is $\leq T(|x|+|y|)^{O(1)}$. (This strengthens a result of Madlener and Otto.)
We also give results that relate the deterministic computational complexity of a representative function and the work of derivations in a finitely generated infinite presentation.

(3) Every deterministic Turing machine with time complexity $O(T)$ is equivalent to a deterministic Turing machine which halts after $O(T)$ steps, no matter what configuration this machine starts in. (This is a complexity version of a theorem by Martin Davis, which plays a key role in the connection between complexity and string rewriting.)

© 1998 Academic Press Limited

Introduction

The word problem of monoids and groups is both an algebraic and a computational problem; this makes it a very interesting object of study. It is particularly interesting

† Supported in part by N.S.F. grant DMS-9203981. Part of this research was carried out during a sabbatical year at the Laboratoire d'Informatique Théorique et Programmation, Institut Blaise Pascal, Université de Paris 6, Paris, France.

‡ E-mail: birget@cse.unl.edu

0747–7171/97/060759 + 35 $25.00/0

© 1998 Academic Press Limited

to characterize the decidability and complexity of the word problem, which are computational properties, by inherently algebraic properties of the (semi)groups. An example of this is the classical Higman Embedding Theorem (Higman, 1961) which states that a finitely generated group has a recursively enumerable word problem iff this group can be embedded into a finitely presented group. A semigroup version of this is given in Murski (1967). The Boone–Higman theorem (Boone and Higman, 1974) and Evans' work (Evans, 1978) are further examples. Madlener and Otto (1985) refined the Higman Embedding Theorem to characterize the Grzegorczyk (primitive recursive) hierarchy from level E_3 onward; see also Bauer and Otto (1984).

In Birget (1998) this theorem was further refined: If the word problem of a finitely generated semigroup S belongs to $\mathrm{NTIME}(T)$ then S is embeddable into a finitely presented semigroup with isoperimetric function $\leq T(.)^2$. Conversely, if a finitely presented semigroup has an isoperimetric function $\leq D(.)$ then all its finitely generated subsemigroups have their word problem in $\mathrm{NTIME}(D)$. Thus in particular, the word problem of S is in the complexity class NP iff S is embeddable in a finitely presented semigroup with polynomially bounded isoperimetric function. A group version of this result is being worked out by the author, jointly with M. Sapir, E, Rips, and A. Ol'shanskii.

In the present paper we use infinite presentations to characterize the complexity of word problems. There are several reasons for working directly with infinite presentations, and there are drawbacks as well. The main drawback is that for infinite presentations the rewrite distance and the rewrite work (defined below) are not algebraic invariants: they depend as much on the presentation as on the monoid or group presented. On the other hand, many monoids and groups have no finite presentation; and even those with a finite presentation do not always have a finite complete (i.e. confluent and terminating) presentation (as proved by Squier (1987); see also Squier and Otto (1987) and Kobayashi (1995)). If one wants to study such (semi)groups by themselves, one needs to look at infinite presentations.

MAIN RESULTS

(1) We show that every monoid with decidable word problem is embeddable in a monoid with a presentation that is *complete* and *whose set of rewrite rules forms a finite-state sequential partial function*; moreover in this presentation, derivations are "efficient" (they have low "derivation work"), compared to the best possible algorithm for the word problem.

Bauer (1981) proved (see the outline in Appendix, Section 2) that if a finitely generated monoid M has a decidable word problem then M is embeddable into a monoid presented by a finite and terminating string rewrite system; this rewrite system is not complete, however, but it is confluent when it starts with a word equivalent to an element of M. This inspired the (still open) question of whether every monoid with decidable word problem is embeddable in a monoid with a finite complete presentation; see Bauer (1981), Otto (1989), Deiss (1993), the discussion at the end of the present paper, and the discussion in Madlener and Otto (1989). However, it is known (see Kobayashi (1995)) that there are monoids with a decidable word problem that do not admit a complete presentation with a regular set of left-sides of the rules; so the embedding is necessary in our theorem.

(2) We show that every monoid with decidable word problem has an infinite presentation whose set of relations has very low complexity; in this presentation, words can

be reduced "efficiently" (in terms of "rewriting work"), compared to the best possible algorithm for computing the reduction.

This improves on a result from Madlener and Otto (1988, Theorems 3.5 and 4.2), who proved the following (reformulated here): Every finitely generated monoid M whose word problem is in E_n (level n in the Grzegorczyk hierarchy) has an infinite presentation $\langle A : R\rangle$ (with finite set of generators A), where the set of relations R is a context-sensitive set (i.e. it belongs to $\mathrm{NSPACE}(n)$), and where the "strong derivation" distance of $\langle A : R\rangle$ is bounded above by a function in E_n. (Here a derivation is "strong" iff it satisfies certain constraints regarding the use of insertion rules $1 \to x$ in the derivation, where 1 is the empty word; see Madlener and Otto (1988).)

We extend this result to more complexity classes, and we obtain a converse for the extended result. We will dispose of the notion of "strong derivation", by measuring the complexity of a derivation in a different way, namely by the "work" of the derivation (defined below), which is more natural than the number of steps in the case of infinite presentations.

(3) The following result on Turing machines plays a key role: Every deterministic Turing machine with time complexity $\mathrm{O}(T)$ is equivalent to a deterministic Turing machine which halts after $\mathrm{O}(T)$ steps, no matter what configuration this machine is started in. This is a complexity version of a theorem from Davis (1956).

This paper owes a lot to the works of Madlener, Otto, and Bauer, cited above. The proof techniques are often similar to the ones in these papers (and in Deiss (1993)). However, we make use of nondeterminism, representative functions, and other notions; this enables us to obtain tighter results for complexity, that have a converse; having a converse implies that the results are optimal (for the given hypotheses). Previous results did not allow converses, and many previous results were not concerned with complexity.

1. Definitions and Preliminary Results

We review some standard definitions, and introduce a few new ones. For the terminology regarding presentations of monoids, monoids and groups by generators and relations, and the word problem, see also Lyndon and Schupp (1977), Sims (1994), Madlener and Otto (1985) and Birget (1998); for definitions and notation concerning algorithms, complexity and nondeterminism, see Hopcroft and Ullman (1979), van Leeuwen (1990).

1.1. PRESENTATIONS

We write $S = \langle A\rangle$ when S is a monoid generated by a set A. The free monoid with free generating set A will be denoted by A^*, the corresponding free semigroup by A^+, and the free group by $\mathrm{FG}(A)$. The empty word of A^* and $\mathrm{FG}(A)$ is denoted by 1. We write $S = \langle A : R\rangle$ when S has a *semigroup presentation* with generating set A and set of relations $R \subseteq A^+ \times A^+$; then S is defined by the semigroup congruence generated by R in A^+. For a *monoid presentation*, (respectively a *group presentation*), the same notation is used, and the situation is similar (replacing A^+ by A^*, (resp. $\mathrm{FG}(A)$), and replacing semigroup congruence by monoid congruence, (resp. group congruence)). The context will indicate whether we intend a presentation to be a semigroup, monoid, or group presentation. A rule $(\alpha, \beta) \in A^* \times A^*$ is usually written as $\alpha \to \beta$. We use the notation $R^{-1} = \{(\beta, \alpha) : (\alpha, \beta) \in R\}$. A presentation $\langle A : R\rangle$ is called *symmetric* iff $R^{-1} = R$.

For any presentation $\langle A : R\rangle$, we call $\langle A : R \cup R^{-1}\rangle$ the corresponding *symmetrized* presentation.

In this paper we will only consider finitely generated (semi)groups and monoids (since otherwise, questions of the decidability and complexity of the word problem are not well defined; see e.g. Birget (1998)).

1.2. DERIVATIONS

Let $\langle A : R\rangle$ be a semigroup (or monoid, or group) presentation, and let $x, y \in A^*$. A *derivation of length* k, from x to y, is a sequence of words $(x =)w_0, w_1, \ldots, w_i, w_{i+1}, \ldots, w_{k-1}, w_k(= y)$, such that w_{i+1} is obtained from w_i by application of one relation of R, for $0 \le i < k$. We write $w_i \to w_{i+1}$ if w_{i+1} is obtained from w_i by application of one rule of R, i.e. w_i and w_{i+1} can be written as $w_i = u\alpha v, w_{i+1} = u\beta v$, for some $u, v \in A^*$, with $(\alpha, \beta) \in R$. We write $x \xrightarrow{*} y$ iff there exists a derivation (of any length, including 0) from x to y.

We write $x \underset{S}{=} y$ (i.e. x and y are congruent) iff there exists a derivation from x to y in the symmetrized presentation $\langle A : R \cup R^{-1}\rangle$ of S. The notation $\underset{S}{=}$ is unambiguous as long as we do not consider more than one congruence on A^+ (defining S) at the same time. The equality sign "=" without any index is reserved for *literal equality* (in the free monoid).

A derivation $(x =)w_0 \to w_1 \to \cdots \to w_i \to w_{i+1} \to \cdots \to w_k(= y)$ is *left-most* iff in each step $w_i = u_i\alpha_i v_i \to w_{i+1} = u_i\beta_i v_i$ the position $|u_i|$ where the rule (α_i, β_i) is applied is minimal (i.e. no rule is applicable to w_i further to the left), for $0 \le i < k$. We call this derivation *greedy* left-most iff the derivation is left-most, and in each step i the rule that is applied has $|\alpha_i|$ maximal (i.e. no rule with a left-side longer than $|\alpha_i|$ is applicable to w_i at the left-most possible position).

In a presentation $\langle A : R\rangle$ we say that R is a *partial function* iff no two rules in R have the same left-side (iff $(\alpha, \beta_1), (\alpha, \beta_2) \in R$ implies $\beta_1 = \beta_2$). Note that *if R is a partial function then from every word there is at most one left-most greedy derivation (of a given derivation length).*

The derivation distance between words x and y, denoted $d_{\langle A:R\rangle}(x, y)$, is the length of the shortest derivation from x to y for the presentation $\langle A : R \cup R^{-1}\rangle$, if $x \underset{S}{=} y$; the distance is undefined when x and y are not equivalent. We view the presentation as being symmetric in order to make the derivation distance-symmetric. The derivation distance depends on the presentation chosen for S, but for finite presentations (of semigroups, monoids or groups) this dependence is only linear in terms of the parameters (this is due to Madlener and Otto (1985); see also Birget (1998), and Theorem 1.2 in the present paper). Therefore we just write $d_S(x, y)$, where S is the semigroup (or monoid or group) presented.

An *isoperimetric function* of $\langle A : R\rangle$ is any function $f : \mathbb{N} \to \mathbb{N}$ such that $f(|x| + |y|) \ge d_{\langle A:R\rangle}(x, y)$ for all x, y with $x \underset{S}{=} y$ (i.e. f is an upper bound on the derivation distance wherever the latter is defined). See e.g. Gersten (1992), Epstein *et al.* (1992), Birget (1998). This geometric terminology will be justified in the next few paragraphs on diagrams. The following fact is straightforward: The word problem of a semigroup or group S is decidable iff S has a total recursive isoperimetric function. (Clearly, the

derivation distance is bounded above by a total recursive function iff the word problem is decidable.)

1.3. DIAGRAMS

The derivation distance has a nice geometric interpretation in terms of van Kampen diagrams. We will concentrate on semigroup diagrams, introduced by Kashintsev (1970) (see Lyndon and Schupp (1977) for the more classical group diagrams, invented by van Kampen in the 1930s). These diagrams are now an essential tool in combinatorial group theory, and they are becoming important in combinatorial semigroup theory too. We follow Remmers (1980) and the textbook by Higgins (1992).

A *semigroup diagram*, with respect to a semigroup presentation $\langle A : R\rangle$, is a planar directed graph embedded in the euclidean plane (endowed with the usual counter-clockwise orientation), whose edges are labeled by elements of A, with the following two properties:

(i) each *cell* (i.e. a bounded face) has a boundary consisting of two maximal nonempty directed simple paths, that have exactly two vertices in common, whose directions (relative to the orientation of the plane) are opposite, and whose two path labels $u, v \in A^+$ satisfy $(u, v) \in R$;
(ii) the graph has exactly one source and one sink, which are both located on the boundary of the unbounded ("external") face.

It follows from the definition that the boundary of the unbounded face consists of two nonempty maximal directed paths, each of which is labeled by a word (say x, respectively y) in A^+; we say that the pair (x, y) labels the outer boundary of the semigroup diagram. We make the convention to call the boundary path which follows a counter-clockwise orientation, when viewed from the outer face (i.e. the path labeled by x), the "*input side*" (called "left side" in Higgins (1992)); the other path y is called the "*output side*" (called "right side" in Higgins (1992)). Note that diagrams are not defined for monoid presentations (when S is a monoid it needs a semigroup presentation, where the empty word is not used; for example, one can add a new letter for the identity element). See Figure 1 (later in the paper).

For symmetrized presentations, the following fundamental relationship holds between van Kampen diagrams and word problems (and derivation distance):
$x \underset{S}{=} y$ *(and the derivation distance satisfies* $d_{\langle A:R\rangle}(x, y) \leq k$*) iff there exists a semigroup diagram whose outer boundary is labeled by* (x, y) *(and whose area, i.e. the number of cells, is* $\leq k$*).*

This was proved by van Kampen for groups (see Lyndon and Schupp (1977)), and by Kashintsev for semigroups (see Remmers (1980) and Higgins (1992)). In the group case one does not count the free-group relations $x^{-1}x = xx^{-1} = 1$ in the derivation distance.

This explains the term "*isoperimetric function*" for any function which is an upper bound on the derivation distance: it is an upper bound on the area of a minimum-area van Kampen diagram for a pair (x, y), as a function of its perimeter $|x| + |y|$.

1.4. DERIVATION WORK

The following notion seems to be new. Let $\langle A : R\rangle$ be a semigroup presentation with A finite (where the set of relations R can be infinite). The *work* of a relation $(u, v) \in R$ is

defined to be $|u|+|v|$. Intuitively, this is the amount of work one has to do to erase u and write v. The work of a derivation $x \xrightarrow{*} y$ is the sum of the works of the relations applied (of course, the same relation will be counted many times if it is applied many times in the derivation). For group presentations, the free-group relations $x^{-1}{\cdot}x = x{\cdot}x^{-1} = 1$ are not counted in the derivation work.

In a symmetric presentation $\langle A : R \cup R^{-1}\rangle$, the work distance $\omega(x, y)$ between two congruent words x and y is the *minimum* work over all derivations from x to y; $\omega(x, y)$ is undefined if x and y are not congruent.

The work of a derivation has a natural interpretation in terms of (group or semigroup) *van Kampen diagrams*. Among the edges of a van Kampen diagram we are particularly interested in *two-face* edges (that belong to the boundaries of two different faces, as opposed to edges that bound only the external face on both sides of the edge), and *internal* edges (that have no edge on the boundary of the external face). See Figure 1 (later in the paper).

PROPOSITION 1.1. *Let K be the van Kampen diagram corresponding to a derivation from x to y. Then the work ω of this derivation is the number e_2 of two-face edges plus the number e_{int} of internal edges of K : $\omega = e_2 + e_{\text{int}}$. Hence in particular, $e_2 \le \omega \le 2e_2$.*

Hence all derivations that have the same van Kampen diagram have the same work.

Moreover, the work of a derivation is big-O of the number of edges of the corresponding van Kampen diagram.

(The proof of Proposition 1.1 is a simple induction on the number of faces of K.)

What makes van Kampen diagrams important is that all derivations that correspond to the same van Kampen diagram K are similar and should not be distinguished. Usually one can pick any convenient derivation for a fixed diagram, rather than studying all derivations. These diagrams are analogous to *parse trees* in context-free language theory.

1.5. CONFLUENT AND TERMINATING PRESENTATIONS

Let $S = \langle A : R\rangle$ be a presentation in which R is asymmetric (that is: if $(\alpha, \beta) \in R$ then $\alpha \neq \beta$ and $(\beta, \alpha) \notin R$). This presentation is *terminating* iff there is no infinite derivation. The presentation is *locally confluent* iff the following holds for all words $w, x, y : x \leftarrow w \rightarrow y$ implies that there is a word z with $x \xrightarrow{*} z \xleftarrow{*} y$. The presentation is confluent iff $x \xleftarrow{*} w \xrightarrow{*} y$ implies that there is a word z with $x \xrightarrow{*} z \xleftarrow{*} y$. A presentation which is both confluent and terminating is called *complete*. For terminating presentations, it is well known that local confluence implies confluence. See for example Sims (1994), Jantzen (1988), or Book and Otto (1993).

A *reduction order* is a total well-order on A^* which is compatible with concatenation (see Sims (1994)). A rewrite system agrees with a strict reduction order $>$ iff the rewrite system satisfies $\alpha > \beta$ for all $(\alpha, \beta) \in R$; in that case the rewrite system is necessarily terminating. More generally, if $\ge$ is a quasi-order (reflexive and transitive) which is compatible with concatenation, and if there are no infinite strictly decreasing chains, then we can again conclude that R is terminating (if R agrees with the corresponding strict quasi-order $>$).

A classical example of a strict reduction order is the strict length-lexicographic order $<_{\text{llex}}$ defined by first ordering words by length, then strictly ordering equally long words lexicographically (see Sims (1994)).

In a complete presentation every word w is equivalent to a unique reduced word, denoted $\mathsf{red}(w)$. The word $\mathsf{red}(w)$ is also called the "reduced" (or "irreducible") *representative* of w.

1.6. REPRESENTATIVES

Even for presentations that are not complete, and more generally, for congruences, one can define representatives of the congruence classes. Suppose that a finitely generated semigroup (or monoid or group) is defined by a congruence $\equiv$ on the free semigroup A^+. (We only describe the case of semigroup presentations from now on, the other cases are similar.)

A *representative function* for $\equiv$ is a function $r : A^+ \to A^+$ such that for all words $x, y \in A^+ : r(x) \equiv x$, and $x \equiv y$ iff $r(x) = r(y)$. The word $r(x)$ is called the representative of the word x (relative to the congruence $\equiv$ and the function r).

For example: If a semigroup S is defined by a congruence $\equiv$ on A^+ and if $\geq$ is a reduction order on A^+ then every congruence class has one $\geq$-minimum element, called the $\geq$-*minimum representative*.

More generally, suppose S is generated by a finite set A_1, and S is embedded into a semigroup H generated by a finite alphabet A with $A_1 \subseteq A$ (where the embedding is induced by the identity function A_1); suppose H has a representative function r. Then we call the restriction of r to words over A_1 a *representative function of S in H*.

1.7. COMPLEXITY

The word problem is closely related to nondeterminism: First, the definition of equivalence of words involves the existential quantifier (there *exists* a derivation ...). Secondly, and more deeply, for a finitely generated semigroup S the nondeterministic complexity of the word problem is polynomially related to the derivation distance, in some finitely presented semigroup into which S is embedded ("Higman-like" embedding theorem for nondeterministic complexity, see Birget (1998)). Unless P = NP (and more generally, nondeterministic time is polynomially bounded by deterministic time), there can be no such result for deterministic time. So in general it is more natural to characterize the nondeterministic complexity of word problems. In order to obtain general results about the deterministic complexity of word problems we will slightly change the problem, as we will see later.

The word problem for $\langle A : R\rangle$ has *nondeterministic time complexity* T (or "belongs to the complexity class $\mathrm{NTIME}(T)$") iff there exists a multitape nondeterministic Turing machine with time complexity function $\mathrm{O}(T)$, accepting the language $\{x\#y : x, y \in A^* \text{ and } x \underset{S}{=} y\}$. So, on input $x\#y$ this Turing machine will have some accepting computation, of length $\mathrm{O}(T(|x| + |y|))$, iff $x \underset{S}{=} y$.

For the algebraic study of word problems it is important that the nondeterministic time complexity is an *algebraic invariant* of finitely generated semigroups or groups (up to a linear change in the parameters); the deterministic time complexity and the space complexity are also invariants. This was proved in Madlener and Otto (1985) (for the Grzegorczyk hierarchy, but the same proof works in general, see Birget (1998)). In particular, the decidability of the word problem is an algebraic invariant (independently

of the finite set of generators and the congruence that defines the group or semigroup); this is well known (see Lyndon and Schupp (1977)).

We will use the parallel complexity class $\mathcal{AC}^0$. This class consists of the problems that can be solved by constant-depth polynomial-size boolean circuits whose gates have finite but unbounded fan-in (with "uniformity conditions"); see van Leeuwen (1990) for a detailed definition. $\mathcal{AC}^0$ is a strict subclass of DSPACE(log), which itself is a subclass of P (deterministic polynomial time). $\mathcal{AC}^0$ corresponds to very low complexity; it does not contain all the finite-state languages, but it does contain the language $\{w\#w : w \in \{a,b\}^*\}$ (so the problem of checking equality of words is in $\mathcal{AC}^0$), and also the language $\{w\#w^{\mathrm{rev}} : w \in \{a,b\}^*\}$ ("palindromes" with a center marker).

We will also use the class DTIME(linear) (also denoted DTIME(n)). Yet another class that will show up is the set of languages that are the *intersection of two deterministic context-free languages*; this is a subclass of DTIME(n). "Deterministic context-free language" will be abbreviated by DCFL. It is an important fact that the set of all accepting computations of a one-tape Turing machine can be represented as the intersection of two DCFLs (see Hopcroft and Ullman (1979), Section 8.6). For a set $R \subseteq A^+ \times A^+$, we will say that "$R$ is the intersection of two DCFLs" iff the language $\{x\#y : (x,y) \in R\}$ is the intersection of two DCFLs (where $\#$ is a new symbol $\notin A$).

Functions T used as complexity bounds are assumed to be *positive* (i.e. $T(n)$ is a positive integer for all $n > 0$), and *superadditive* (i.e. $T(n+m) \geq T(n) + T(m)$, for all n, m). These assumptions imply that T is also strictly increasing. All the functions that are traditionally used as complexity bounds in the analysis of algorithms satisfy these assumptions.

1.8. COMPLEXITY OF PARTIAL FUNCTIONS

For a (partial) function $f : A^+ \to A^+$ the deterministic time-complexity can be defined in various (nonequivalent) ways. The first definition is fairly standard (see van Leeuwen (1990)); the other ones are less standard.

(1) A partial function f belongs to the class F-DTIME(T) (or has *functional* deterministic time-complexity T) iff there is a deterministic Turing machine which, for every input $x \in A^+$, outputs $f(x)$ (or goes to a reject state if $f(x)$ is undefined) after a computation of duration $O(T(|x|))$. So here T uses only the input-length $|x|$ as a parameter.

An important special case is the class FP, which is the union of all the classes F-DTIME(p) as p ranges over the set of all polynomials.

(2) A partial function f belongs to the class I/O-DTIME(T) (or has *input–output* deterministic time-complexity T) iff there is a deterministic Turing machine which, for every input $x \in A^+$, outputs $f(x)$ after a computation of duration $O(T(|x| + |f(x)|))$, or goes to a reject state after $O(T(|x|))$ steps if $f(x)$ is undefined. The difference with (1) is that here T uses the combined input–output length $|x| + |f(x)|$ as a parameter.

In particular, I/O-DTIME(linear) is defined as above, by taking $T(|x| + |f(x)|) = |x| + |f(x)|$ when x is in the domain of f; the time is $O(|x|)$ when x is outside the domain of f.

(3) A partial function f belongs to the class DTIME(T) (or has deterministic time-complexity T, *as a language*) iff the language $\{x\#f(x) : x \in A^+\}$ belongs to DTIME(T), where $\#$ is a new letter $\notin A$. Since here the input is $x\#f(x)$, the parameter of T is $|x\#f(x)|$. The difference between this and (2) is that now the Turing machine has both x and $f(x)$ available at the beginning of its computation (and works deterministically

when it knows both; the machine "verifies" that the given word $f(x)$ is indeed the image of x).

1.9. COMPLEXITY OF REPRESENTATIVE FUNCTIONS

If S has a representative function in F-DTIME(T), with respect to some congruence $\equiv$, then the word problem of S is in DTIME(T) (since $x \equiv y$ iff $r(x) = r(y)$). In the study of deterministic complexity we will replace the word problem by the problem of computing a representative function $r(x)$.

Note that a finitely generated semigroup S has a decidable word problem iff S admits a representative function which is total recursive. (The implication $\Leftarrow$ is trivial; for the implication $\Rightarrow$, observe that the $\leq_{\text{llex}}$-minimum representative function is total recursive when the word problem is decidable.)

Interestingly, the existence of representative functions with certain deterministic functional and input–output complexities is an algebraic invariant of a finitely generated semigroup; this extends the invariance of the complexity of the word problem, proved in Madlener and Otto (1985).

We will make the following (standard) use of the big-O notation: If T is a function then $T(\mathrm{O}(.))$ denotes the class of all functions $f : \mathbb{N} \to \mathbb{N}$ such that $f(n) \leq T(c \cdot n)$ for all n (where c is a "constant" which does not depend on n but depends on f). Accordingly, F-DTIME($T(\mathrm{O}(.))$) is the union of all F-DTIME(f) as f ranges over $T(\mathrm{O}(.))$.

THEOREM 1.1. ALGEBRAIC INVARIANCE OF REPRESENTATIVE FUNCTIONS AND OF THEIR COMPLEXITY. *Let S be a finitely generated semigroup which is defined by two semigroup congruences $\equiv_1$ (over a finite alphabet A_1) and $\equiv_2$ (over a finite alphabet A_2). Suppose that r_1 is a representative function for $\equiv_1$ and suppose that r_1 belongs to the complexity class* F-DTIME(T) *(or* I/O-DTIME(T)*). We assume that the function T is increasing and satisfies $T(n) \geq n$ for all n.*

Then there exists a representative function r_2 for $\equiv_2$ such that r_2 belongs to F-DTIME($T(O(.))$)*, respectively* I/O-DTIME($T(O(.))$)*. Moreover, if the function l_1 is an upper bound on the length of r_1 (i.e. $|r_1(x)| \leq l_1(|x|)$ for all $x \in A_1^+$), then $|r_2(y)| \leq c_2 \cdot l_1(c_1.|y|)$ for all $y \in A_2^+$ (where c_1 and c_2 are constants).*

PROOF. Since both $\equiv_1$ and $\equiv_2$ define semigroups isomorphic to S, there are isomorphisms $i_1 : A_1^+/_{\equiv_1} \to S$ and $i_2 : A_2^+/_{\equiv_2} \to S$. We denote the equivalence class of x for $\equiv_i$ by $[x]_i$ for $i = 1, 2$. For every letter $a_1 \in A_1$ we choose a word $\varphi_2(a_1) \in A_2^+$ such that a_1 and $\varphi_2(a_1)$ are "equivalent in S", i.e. $i_1([a_1]_1) = i_2([\varphi_2(a_1)]_2) \in S$. Similarly, for every letter $a_2 \in A_2$ we choose $\varphi_1(a_2) \in A_1^+$ such that $i_2([a_2]_2) = i_1([\varphi_1(a_2)]_1) \in S$. We extend the functions φ_1 and φ_2 to homomorphisms, which we also call φ_1 (from A_2^+ to A_1^+), respectively φ_2 (from A_1^+ to A_2^+).

We now define the representative function r_2 of $\equiv_2$ as follows:

$$r_2(y) = \varphi_2 r_1 \varphi_1(y), \text{ for all } y \in A_2^+.$$

From this definition we immediately conclude that $|r_2(y)| \leq c_2 \cdot l_1(c_1 \cdot |y|)$, where l_1 is as in the theorem; the constants c_1 and c_2 are given by $c_1 = \max\{|\varphi_1(a_2)| : a_2 \in A_2\}$, and $c_2 = \max\{|\varphi_2(a_1)| : a_1 \in A_1\}$. Let us prove next that r_2 is a representative function for $\equiv_2$.

Proof that $r_2(y) \equiv_2 y$ for all $y \in A_2^+$: By definition of φ_2 we have $(i_2([r_2(y)]_2) =)$

$i_2([\varphi_2 r_1 \varphi_1(y)]_2) = i_1([r_1\varphi_1(y)]_1)$; since r_1 is a representative function we also have $i_1([r_1\varphi_1(y)]_1) = i_1([\varphi_1(y)]_1)$; and by the definition of φ_1 we have $i_1([\varphi_1(y)]_1) = i_2([y]_2)$. Altogether this yields $i_2([r_2(y)]_2) = i_2([y]_2)$, hence $r_2(y) \equiv_2 y$.

Proof that $r_2(x) = r_2(y)$ implies $x \equiv_2 y$: This follows from the property we just proved, by transitivity of $\equiv_2$.

Proof that $x \equiv_2 y$ implies $r_2(x) = r_2(y)$: From $x \equiv_2 y$ it follows that $\varphi_1(x) \equiv_1 \varphi_1(y)$, hence $r_1\varphi_1(x) = r_1\varphi_1(y)$; by applying φ_2 we obtain $(r_2(x) =)\varphi_2 r_1\varphi_1(x) = \varphi_2 r_1\varphi_1(y)(= r_2(y))$.

Proof that r_2 belongs to F-DTime(T(O(.))), respectively to I/O-DTime(T(O(.))): Given a deterministic Turing machine M_1 computing r_1 in time T, we construct a deterministic Turing machine M_2 for r_2 as follows: On input x_2, M_2 first computes $\varphi_1(x_2)$ (in time $\le c_1 \cdot |x_2|$). Then it simulates M_1 on $\varphi_1(x_2)$ to compute $r_1\varphi_1(x_2)$. In the "F-case" this takes time $\le T(|\varphi_1(x_2)|) \le T(c_1 \cdot |x_2|)$. In the "I/O-case" this takes time $\le T(|\varphi_1(x_2)| + |r_1\varphi_1(x_2)|) \le T(c_1 \cdot |x_2| + |r_2(x_2)|)$; the second "$\le$" holds since $|r_1\varphi_1(x_2)| \le |\varphi_2 r_1\varphi_1(x_2)|(= |r_2(x_2)|)$ because φ_2 is non-erasing. Finally, M_2 computes $\varphi_2 r_1\varphi_1(x_2)(= r_2(x_2))$, which takes time $|\varphi_2 r_1\varphi_1(x_2)|$; in the "I/O-case" we simply observe that this time is $= |r_2(x_2)|$; in the "F-case" we have $|\varphi_2 r_1\varphi_1(x_2)| \le c_2 \cdot |r_1\varphi_1(x_2)| \le c_2 \cdot T(c_1 \cdot |x_2|)$, since here the output-length is bounded by the time complexity.□

1.10. universally halting Turing machines

The following result from Davis (1956) about Turing machines is important for word problems, and played a critical role in Madlener and Otto (1985), Bauer and Otto (1984), and Bauer (1981): If a deterministic Turing machine M always eventually halts when started in an input configuration, then M is equivalent to a deterministic Turing machine M′ that always eventually halts, no matter what configuration M′ starts in. Such a Turing machine is said to be *universally halting*.

An *input configuration* of a Turing machine is of the form $¢q_0w\$$, with all other tapes blank, where w is a word over the input alphabet; see Appendix A1 for more details about Turing machines.

By definition, two Turing machines are equivalent iff they accept the same language (in the case of acceptors), or have the same input–output function (in the case of transducers).

We introduce a stronger form of the theorem by Davis, by strengthening the proof in such a way that *time-complexity is preserved*.

Theorem 1.2. *Let* M *be a deterministic Turing machine with time-complexity T, i.e.* M *halts after $\le T(|w|)$ steps when started on any input configuration* $¢\mathrm{q}_0w\$$. *Then* M *is equivalent to a deterministic Turing machine* M′ *that always halts after* O(T(|C|)) *steps, no matter what configuration* C *the machine* M′ *starts in.*

M *is also equivalent to a deterministic* one-tape *Turing machine* M'_1 *that always halts after* O($T(|\mathrm{C}_1|)^2$) *steps, no matter what configuration* C_1 *the machine* M'_1 *starts in. Moreover, in a rejecting configuration the tape is empty. And in an accepting computation with input x and output y, every configuration is at least as long as* $3 + \min\{|x|, |y|\}$.

In addition M′ has the following property: All tapes (except the output tape) are empty at the end of every computation.

The proof of Theorem 1.3 is given in Appendix A3.

1.11. RATIONAL PRESENTATIONS

A presentation $\langle A : R \rangle$, with A finite, is rational iff R is a rational subset of $A^* \times A^*$.

A standard reference on rational subsets of monoids and on finite transducers is Berstel (1979); let us give a short and incomplete description of these notions now. By definition, a subset L of $A^* \times A^*$ is rational iff L has a rational (also called "regular") expression over the generating set $A \times \{1\} \cup \{1\} \times A$ of $A^* \times A^*$. Equivalently, R is accepted by a nondeterministic finite transducer: $(x, y) \in R$ iff on input x the transducer has some accepting computation with output y. We say that an output y is "valid" iff the computation that produces y as an output ends in an accept state. A nondeterministic finite transducer is a machine $(\mathrm{Q}, A, \delta, q_0, \mathrm{F})$ where Q is the set of states, A is the input–output alphabet, $\mathrm{q}_0 (\in \mathrm{Q})$ is the start state, $\mathrm{F} (\subseteq \mathrm{Q})$ is the set of accept states, and $\delta \subseteq \mathrm{Q} \times A^* \times \mathrm{Q} \times A^*$ is the transition relation; Q, A, and δ are assumed finite.

As a special case of rational subsets of $A^* \times A^*$ we have the *finite-state sequential (partial) functions*; those are the input–output (partial) functions of finite transducers that are deterministic (in the strong sense: the input x determines at most one computation); they are also called "sequential machines", or "Mealy machines". See Section IV.2 in Berstel (1979). Compare also with Hopcroft and Ullman (1979), Section 2.7; our sequential machines are a little more general than those in Hopcroft and Ullman (1979) since we also use accept states; an output is only valid if the state reached is an accept state.

Note that if R is a rational subset of $A^* \times A^*$ then the first projection of R (called "the domain" or "the set of left-sides" of R) is a regular language; in the literature on string rewriting, a rewrite system is often called "regular" if the set of left-sides of R is a regular language. This is a weaker constraint than rationality of R.

2. Embedding into Complete Rational Presentations

In this section we give a Higman-like embedding theorem for semigroups, in order to characterize *deterministic* time-complexity (and decidability). It is an open question whether every finitely generated semigroup with decidable word problem is embeddable into a complete finite string rewrite system; see the beginning of the Introduction and Section 4. In this section we deal only with *semigroup presentations*.

Recall that we assume that complexity bounds T are positive and superadditive (and total recursive).

THEOREM 2.1. *(a) Assume a finitely generated semigroup S is embeddable into a finitely generated semigroup H_1 with the following property: H_1 has a representative function r_1 which belongs to* F-DTIME(T)*, and which satisfies $|r_1(x)| \leq \mathrm{O}(|x|)$, for all words x. (S and H_1 could be the same.) Then S can be embedded into a semigroup H which has a complete presentation $\langle A : R \rangle$, where A is finite and R is a rational partial function, and where every derivation $x \xrightarrow{*} \mathsf{red}(x)$ has work $\leq |x| \cdot T(\mathrm{O}(|x|))^2$; in addition, $|\mathsf{red}(x)| \leq \mathrm{O}(|x|)$.*

Moreover, S can be embedded into a finitely generated semigroup H which has a complete *presentation $\langle A : R \rangle$, where A is finite and R is a finite-state sequential partial function, and where every derivation $x \xrightarrow{*} \mathsf{red}(x)$ has work $\leq |x| \cdot T(\mathrm{O}(|x|))^3$; in addition, $|\mathsf{red}(x)| \leq \mathrm{O}(|x|)$.*

(b) Conversely, suppose a finitely generated semigroup S can be embedded into a finitely generated semigroup H (which might be S itself), which has a complete and rational

presentation in which every derivation $x \xrightarrow{*} \mathsf{red}(x)$ *has work* $\leq W(|x|)$*; we assume that the bound* $W(.)$ *is a total recursive function satisfying* $W(n) \geq n$*. Then* H *has a representative function with complexity* F-DTIME(W)*, and* S *has a representative function in* H*, with complexity* F-DTIME($W(\mathrm{O}(.))$).

Observe that the assumptions on r_1 in Theorem 2.1(a) are algebraically invariant, i.e. they are an intrinsic property of S (by Theorem 1.2).

COROLLARY 2.1. (DECIDABLE WORD PROBLEM) *The following are equivalent for a finitely generated semigroup* S*:*

- S *has a decidable word problem;*
- S *is embeddable into a finitely generated semigroup with a complete rational presentation;*
- S *is embeddable into a finitely generated semigroup with a complete presentation* $\langle A : R\rangle$ *whose set of relations* R *is a finite-state sequential partial function.*

The assumption in Theorem 2.1 that the length of r_1 be linearly bounded is often satisfied in the examples considered in the literature (see e.g. Bauer and Otto (1984), Sims (1994)).

More generally, in the proof of the theorem we shall see (Lemma 2.4):

If there is a superadditive function l_1 *such that* $|r_1(x)| \leq l_1(|x|)$*, then the work in Theorem 2.1(a) is* $\leq |x| \cdot T(l_1(|x|))^2$ *or* $|x| \cdot T(l_1(|x|))^3$*; moreover* $|\mathsf{red}(x)| \leq l_1(|x|) \leq T(|x|)$.

Hence, when T ranges over all polynomials the restriction "$|r_1(x)| \leq \mathrm{O}(|x|)$" can be dropped. So we have the following characterization of deterministic polynomial time complexity of representative functions (the class FP):

COROLLARY 2.2. (DETERMINISTIC POLYNOMIAL TIME) *For any finitely generated semigroup* S *the following are equivalent:*

- S *is embeddable into a finitely generated semigroup* H_1 *that has a representative belonging to FP;*
- S *is embeddable into a complete presentation* $\langle A : R\rangle$ *with* A *finite and* R *a rational subset of* $A^+ \times A^+$*, such that every derivation* $x \xrightarrow{*} \mathsf{red}(x)$ *in* $\langle A : R\rangle$ *has polynomially bounded work.*

PROOF OF THEOREM 2.1(A). If H_1 embeds into H then S automatically embeds into H. Let H_1 be defined by a congruence $\equiv_1$ on A_1^+ where A_1 is a finite alphabet, and let r_1 be a representative function for $\equiv_1$. Let $\mathrm{M} = (\mathrm{Q}, \Gamma, ¢, \$, \delta, \mathrm{q}_0, \mathrm{q}_\mathrm{f})$ be a deterministic Turing machine computing $r_1(x)$ on input x in time $\leq T(|x|)$. See Appendix A1 for more details about Turing machines. The start configuration on input x is $¢\mathrm{q}_0 x\$$ (with all other tapes blank); at the end of the computation of M on input x the configuration is $¢\mathrm{q}_\mathrm{f} r_1(x)\$$ (with all other tapes blank). By Theorem 1.3, we may assume that M is *universally halting*, and that the time-complexity bound $\mathrm{O}(T)$ always holds, no matter what configuration M starts in.

We first prove the *sequential function* part of the theorem. We convert the multi-tape Turing machine M into a one-tape Turing machine $\mathrm{M}_1 = (\mathrm{Q}_1, \Gamma_1, ¢, \$, \delta_1, \mathrm{q}_{0,1}, \mathrm{q}_{\mathrm{f},1})$, according to a classical construction (Appendix A2). By Theorem 1.3 we may assume that M_1 is universally halting, and the time-complexity of M_1 is $\leq \mathrm{O}(T(|\mathrm{C}_1|)^2)$, no matter what the initial configuration C_1 of M_1 is.

We embed H_1 into the semigroup H with the following presentation $\langle A : R \rangle$:

> Generators: $A = \mathrm{Q}_1 \cup \Gamma_1 \cup \{¢, \$\}$.
> Relations: $R = \{\mathrm{C} \to \mathrm{C}' : \mathrm{C}$ and C' are configurations of M_1 and C' is reachable from C by one transition of $\mathrm{M}_1\} \cup \{¢\mathrm{q}_{\mathrm{f},1}x\$¢\mathrm{q}_{\mathrm{f},1}y\$ \to ¢\mathrm{q}_{0,1}xy\$: x, y \in A_1^+\}$.

The embedding of H_1 into H is induced by the following map on the generators:

$$a \in A_1 \to ¢\mathrm{q}_{0,1}a\$ \in A^+.$$

The work of a rule $\mathrm{C} \to \mathrm{C}'$ is $|\mathrm{C}| + |\mathrm{C}'| \leq 2.|\mathrm{C}| + 1$, since C' has at most one more letter than C.

The presentation of H is intuitive, and based on the same general idea as in Murski (1967), Bauer (1981, 1985) and Deiss (1993), but much simpler. The embedding is similar to the one in Murski (1967) and Birget (1998), and to the representation map in Deiss (1993) (and simpler than in Bauer (1985)). The difference is in the result itself and in the proof (where computational complexity and work distance play a major role), in the proof of a converse, in the infinity of the presentation (which simplifies the presentation but leads to rationality and complexity issues), and in the Turing machine (universally halting with the same time complexity). Also, in Deiss (1993) and Bauer (1981) the embedding is the identity on the generators, which is not the case in our construction; however, this property could easily be introduced in our construction (without changing other properties), by adding new generators into H via a Tietze transformation (see Lemma 2.5).

In the following claims we prove all the properties stated in the theorem. To simplify the notation, from now on we will write q_0 and q_f instead of $\mathrm{q}_{0,1}$ and $\mathrm{q}_{\mathrm{f},1}$; it will be clear from the context that we are talking about states of M_1.

Claim. The set of relations R of H is a *finite-state sequential* partial function.

Proof. Recall that a sequential machine has accept states; an output is only valid if it is produced by a computation that ends in an accept state.

The set $\{(¢\mathrm{q}_\mathrm{f}x\$¢\mathrm{q}_\mathrm{f}y\$, ¢\mathrm{q}_0xy\$) : x, y \in A_1^+\}$ is obviously a finite-state sequential function. For the set $\{(\mathrm{C}, \mathrm{C}') : \mathrm{C}$ and C' are configurations of M_1 and C' is reachable from C by one transition of $\mathrm{M}_1\}$, we have to look at each of the four different kinds of transitions that a one-tape Turing machine is capable of (see Appendix A1). A configuration C has the form $¢uqv\$$. A deterministic sequential machine can be constructed, which on input C outputs C' (by remembering the last few letters of C, and applying a rule). The details are straightforward.□

Claim. The rewrite system R is complete. Thus, every word $x \in A^+$ has a unique reduced representative $\mathsf{red}(x)$ relative to R.

PROOF. Recall that we assume that M_1 is deterministic and that its accept state q_f is a sink (Appendix A1).

(1) We first show *local confluence*, by the straightforward method consisting of showing that when two left-sides of rules of R overlap in a word then the two words derived this way have a common descendant (see Sims (1994); Jantzen (1988) and Book and Otto (1993)).

Two rules of the form $¢u_1q_1v_1\$ \to ¢u'_1q'_1v'_1\$$ and $¢u_2q_2v_2\$ \to ¢u'_2q'_2v'_2\$$ cannot overlap, unless their left-sides are identical; but in that case, the right-sides are also identical, by the determinism of M_1.

Two rules, one of the form $¢u_1q_1v_1\$ \to ¢u'_1q'_1v'_1\$$ and the other of the form $¢q_fx\$¢q_fy\$ \to ¢q_0xy\$$ cannot overlap, since q_f is a sink state.

The only remaining possibility is an overlap of the following form (where $x, y, z \in A_1^+$) : $¢q_0xy\$¢q_fz\$ \leftarrow ¢q_fx\$¢q_fy\$¢q_fz\$ \to ¢q_fx\$¢q_0yz\$$.

Then, for the left branch of the derivation there exists a derivation $¢q_0xy\$¢q_fz\$ \xrightarrow{*} ¢q_fr_1(xy)\$¢q_fz\$ \to ¢q_0r_1(xy)\cdot z\$ \xrightarrow{*} ¢q_fr_1(r_1(xy)\cdot z)\$$. Similarly, for the right branch of the derivation there exists a derivation $¢q_fx\$¢q_0yz\$ \xrightarrow{*} ¢q_fx\$¢q_fr_1(yz)\$ \to ¢q_0x\cdot r_1(yz)\$ \xrightarrow{*} ¢q_fr_1(x\cdot r_1\,(yz))\$$. The result is the same in both cases since $r_1(r_1(xy)\cdot z) = r_1(x\cdot r_1(yz)) = r_1(xyz)$.

(2) Next we show that R is *terminating*. When a derivation starts from a word x, there will be less than $|x|$ applications of "restart rules" of the form $¢q_fx\$¢q_fy\$ \to ¢q_0xy\$$. Indeed, a restart rule decreases the number of occurrences of ¢ and \$ in a word (and no rule increases these numbers).

Before, between, and after applications of restart rules, the derivation uses "transition rules" of the form $C \to C'$ where C and C′ are configurations of M_1. Since M_1 is universally halting, there cannot be an infinite sequence of applications of transition rules.□

The upper bound on the derivation work given in Theorem 2.1(a) will follow from the more general Lemma below.

LEMMA 2.2. *Suppose the representative function r_1 of H_1 belongs to* F-DTIME(T) *and satisfies $|r_1(x)| \le l_1(|x|)$, where T and l_1 are superadditive and positive. Then every derivation $x \xrightarrow{*} \mathsf{red}(x)$ in H has work $\le O(|x|\cdot T(l_1(|x|))^3)$; moreover $|\mathsf{red}(x)| \le l_1(|x|)$.*

PROOF OF LEMMA 2.4. Consider any derivation $x \xrightarrow{*} \mathsf{red}(x)$, with work ω. We define the following factorization of x: first we pick all subsegments of x of the form $¢\alpha\$$ that are configurations of M_1 (in particular, α does not contain ¢ nor \$); after marking these subsegments as factors, we pick the remaining maximal subsegments as factors. Since words of the form $¢\alpha\$$ (where α does not contain ¢ nor \$) never overlap, this factorization of x is unique. Factors of the first type are called "configuration factors", and factors of the second type are called "junk factors".

We will prove the Lemma by induction on the length of x. When x has length 0, 1, or 2, the Lemma is obvious. Suppose now $|x| > 2$.

From the definition of the rewrite system R one sees that when a rule is applied to a word, no junk factor is modified. Similarly, a rejecting configuration factor (i.e. a configuration to which no transition of M_1 can be applied, and which is not an accepting configuration) can never be modified. Thus, if x can be factored as $x = v_1zv_2$ where z is

a junk factor of x or a rejecting configuration, and v_1, v_2 are products of other factors of x (or empty), then $\mathsf{red}(x) = \mathsf{red}(v_1)z\mathsf{red}(v_2)$. By induction, the work of our derivation $x \xrightarrow{*} \mathsf{red}(x)$ is $\omega = \omega_1 + \omega_2 \leq c \cdot (|v_1| \cdot T(l_1(|v_1|))^3 + |v_2| \cdot T(l_1(|v_2|))^3)$, where ω_1 (resp. ω_2) is the work involved in deriving $\mathsf{red}(v_1)$ from v_1 (resp. $\mathsf{red}(v_2)$ from v_2) as part of our derivation $x \xrightarrow{*} \mathsf{red}(x)$, and where $c > 0$ is a constant. Hence, since the functions $l_1(.)$ and $n \cdot T(n)^3$ are superadditive, $\omega \leq c \cdot (|v_1 v_2| \cdot T(l_1(|v_1 v_2|))^3) \leq c \cdot (|x| \cdot T(l_1(|x|))^3)$. For the length we have $|\mathsf{red}(x)| = |\mathsf{red}(v_1)| + |z| + |\mathsf{red}(v_2)|$, hence by induction, $|\mathsf{red}(x)| \leq l_1(|v_1|) + |z| + l_1(|v_2|) \leq l_1(|x|)$, where the latter inequality uses superadditivity and positivity of $l_1(.)$.

We are left with the case where x is a product of configuration factors, none of which is rejecting. If one of these factors is a configuration which, according to the transitions of M_1, leads to a rejecting configuration, then we are again in the previous case: $\mathsf{red}(x) = \mathsf{red}(v_1)z\mathsf{red}(v_2)$, where z is a reject configuration, so $|z|$ is a constant (in a reject configuration the tape is empty, by Theorem 1.3). The rest of the reasoning is similar.

Finally, we consider the case where all factors of x are configurations that lead to accept configurations: $x = \mathrm{C}_1 \ldots \mathrm{C}_k$, where each C_i is a configuration that is eventually rewritten to $¢\mathrm{q_f}w_i\$$ during our derivation (for some reduced word $w_i = r_1(w_i) \in A_1^+, 1 \leq i \leq k$). Some neighboring factors $¢\mathrm{q_f}w_i\$¢\mathrm{q_f}w_{i+1}\$$ are eventually rewritten to $¢\mathrm{q_0}w_i w_{i+1}\$$, then rules corresponding to transitions of M_1 are applied to this, etc. Thus in the end, $\mathsf{red}(x) = ¢\mathrm{q_f}r_1(w_1 \ldots w_k)\$$. So, for the length we have $|\mathsf{red}(x)| = 3 + |r_1(w_1 \ldots w_k)| \leq 3 + l_1(|w_1| + \ldots + |w_k|) \leq 3 + l_1(|\mathrm{C}_1| - 3 + \ldots + |C_k| - 3)$; the last inequality holds since $|¢\mathrm{q_f}w_i\$| \leq |\mathrm{C}_i|$ by Theorem 1.3. And by superadditivity (and since $k \geq 1$), $3 + l_1(|\mathrm{C}_1| - 3 + \ldots + |C_k| - 3) = 3 + l_1(|x| - 3k) \leq l_1(|x|)$. The work of our derivation $x \xrightarrow{*} \mathsf{red}(x)$ is bounded as follows in this case. "Restart rules" of the form $¢\mathrm{q_f}w'\$¢\mathrm{q_f}w''\$ \to ¢\mathrm{q_0}w'w''\$$ are applied less than $|x|$ times (since such a rule decreases the number of occurrences of ¢ and \$). So we still need to show that between applications of restart rules the work is $\leq c' \cdot T(l_1(|x|))^3$, for some constant c'.

Every time an accepting configuration is reached, this configuration will have the form $¢\mathrm{q_f}r_1(w_i \ldots w_j)\$$ (with $1 \leq i \leq j \leq k$), and we have $|r_1(w_i \ldots w_j)| \leq l_1(|w_i \ldots w_j|)$. Thus, when a restart rule is applied we always obtain a start configuration of length $\leq l_1(|x|)$. Therefore the subsequent work on this factor (until the next application of a restart rule) corresponds to the computation of M_1 on this factor, and takes time $\leq c_1 T(l_1(|x|))^2$ and space $\leq c_2 T(l_1(|x|))$, hence work $\leq c_1 c_2 T(l_1(|x|))^3$, for constants c_1, c_2.

So the total work of our derivation $x \xrightarrow{*} \mathsf{red}(x)$ in this case is $\mathrm{O}(|x| \cdot T(l_1(|x|))^3)$.□

Claim. The function $e : x \in A_1^+ \to ¢\mathrm{q_0}x\$ \in A^+$ induces an *embedding* (i.e. an injective homomorphism) of H_1 into H.

Proof. First we show that e induces a function from H_1 to H, by showing that if $x, y \in A_1^+$ are equivalent in H_1 then $e(x)$ and $e(y)$ are equivalent in H. Indeed, if x and y are equivalent in H_1 then $r_1(x) = r_1(y)$; moreover, transitions of M_1 give derivations $e(x) = ¢\mathrm{q_0}x\$ \xrightarrow{*} ¢\mathrm{q_f}r_1(x)\$ = ¢\mathrm{q_f}r_1(y)\$ \xleftarrow{*} ¢\mathrm{q_0}y\$ = e(y)$.

To show that e is a homomorphism we show that for all words $x, y \in A_1^+$, the word $¢\mathrm{q_0}x\$¢\mathrm{q_0}y\$$ is equivalent (in H) to the word $¢\mathrm{q_0}xy\$$. Indeed, in H we have the derivations $¢\mathrm{q_0}x\$¢\mathrm{q_0}y\$ \xrightarrow{*} ¢\mathrm{q_f}r_1(x)\$¢\mathrm{q_f}r_1(y)\$ \to ¢\mathrm{q_0}r_1(x) \cdot r_1(y)\$ \xrightarrow{*} ¢\mathrm{q_f}r_1(r_1(x) \cdot r_1(y))\$$, and this last word is identical to $¢\mathrm{q_f}r_1(xy)\$$ since r_1 is a representative function; on the other hand, we also have $¢\mathrm{q_0}xy\$ \xrightarrow{*} ¢\mathrm{q_f}r_1(xy)\$$.

To show injectiveness of the homomorphism, we show that if $e(x)$ and $e(y)$ are equivalent in H then x and y are equivalent in H_1. Indeed, if $¢q_0x\$$ and $¢q_0y\$$ are equivalent in H then by the completeness of the presentation of H (which we already proved), there exists $w \in A^+$ such that $¢q_0x\$ \xrightarrow{*} w \xleftarrow{*} ¢q_0y\$$, where in addition, w is reduced. On the other hand, $¢q_0x\$ \xrightarrow{*} ¢q_fr_1(x)\$$ and $¢q_fr_1(x)\$$ is also reduced relative to our presentation of H. Thus $w = ¢q_fr_1(x)\$$; similarly, $w = ¢q_fr_1(y)\$$. Thus, $¢q_fr_1(x)\$$ and $¢q_fr_1(y)\$$ are identical words, so $r_1(x) = r_1(y)$, hence x and y are equivalent in H_1.□

This completes the proof of the sequential-function part of Theorem 2.1.

We prove next that if one changes the presentation one can decrease the work of all derivations $x \xrightarrow{*} \mathsf{red}(x)$ to be $\leq |x| \cdot T(O(|x|))^2$. However this bears a cost, as the set of relations is now a *rational partial function* (instead of the more special finite-state sequential function).

We embed H_1 into a semigroup $H = \langle A : R\rangle$; we keep the same notation as before, although H and R are now different. This will not cause any confusion since we will no longer refer to the notation of the sequential-function case.

Recall that the representative function r_1 of H_1 is computed by a k-tape universally halting deterministic Turing machine $M = (Q, \Gamma, ¢, \$, \delta, q_0, q_f)$, with time complexity $O(T)$. Let $M_1 = (Q_1, \Gamma_1, ¢, \$, \delta_1, q_{1,0}, q_{1,f})$ be the corresponding one-tape machine, as in Appendix A2.

The semigroup H has the following presentation:

> Generators: $A = Q_1 \cup \Gamma_1 \cup \{¢, \$\}$.
> Relations: $R = \{C \to C' : C$ and C' are configurations of M_1 that represent configurations of M, and C' is reachable from C by the simulation of one transition of M (i.e. by one simulating double sweep of $M_1)\} \cup \{¢q_{f,1}x\$¢q_{f,1}y\$ \to ¢q_{0,1}xy\$: x, y \in A_1^+\}$.

The embedding of H_1 into H is induced by the following map on the generators:

$$a \in A_1 \to ¢q_{0,1}a\$ \in A^+.$$

To simplify the notation, we will again write q_0 and q_f instead of $q_{0,1}$ and $q_{f,1}$; it will be clear from the context that we are talking about states of M_1.

Claim. The set of relations R is a *rational* partial function.

Proof. The set $\{(¢q_fx\$¢q_fy\$, ¢q_0xy\$) : x, y \in A_1^+\} \subseteq A^* \times A^*$ is rational. Indeed, it is the homomorphic image of the regular language $¢\#A_1^+£A_1^+\$$ (over the alphabet $A \cup \{£, \#\}$, where $£, \#$ are new symbols not in A), under the homomorphism which maps $a \in A$ to (a, a), $\#$ to $(q_{f,1}, q_{0,1})$, and $£$ to $(\$¢q_{f,1}, 1)$. (Recall that 1 denotes the empty word.)

Since the union of rational sets is rational, we only need to show that the set $\{(C, C') : C$ and C' are configurations of M_1 that $\ldots\}$ is rational. We construct a nondeterministic transducer for this set as follows:

The transducer will reject any input which is not a configuration of M_1 representing a configuration of M; no output is produced in that case. Clearly, the words over $Q_1 \cup \Gamma_1 \cup \{¢, \$\}$ that represent configurations of M form a regular language (see the multi-tape to one-tape conversion).

On an input C which represents a configuration of M, the transducer first guesses a transition of M, then reads C to check whether the guessed transition is applicable, applies the transition at the same time, and outputs the new configuration. If C is not a configuration of M or the guessed transition is not applicable to C, the transducer goes to a reject state; thus the output will not be valid. So, for this transducer there exists at least one accepting computation in which it produces the next configuration C′ of M, if C is a configuration of M; and the transducer never produces a wrong next configuration.□

Claim. The rewrite system R is complete.

Proof. The proof is the same as in the case where R is a sequential function.□

Claim. In the rewrite system R, every derivation $x \xrightarrow{*} \mathsf{red}(x)$ has $work \leq |x| \cdot T(\mathrm{O}(|x|))^2$; moreover $|\mathsf{red}(x)| \leq l_1(|x|)$.

Proof. The proof is almost the same as in the case where R is a sequential function. The only difference is that M has time-complexity $T(.)$ whereas M_1 has time-complexity $T(.)^2$. The new rewrite system R carries out one transition of M in one rule, whereas the previous rewrite system carried out one transition of M_1 in one rule. This leads to a derivation work $\leq |x| \cdot T(\mathrm{O}(|x|))^2$ instead of $|x| \cdot T(\mathrm{O}(|x|))^3$.□

Claim. The function $e : x \in A_1^+ \rightarrow \mathrm{¢q}_0 x\$ \in A^+$ induces an embedding (i.e. an injective homomorphism) of H_1 into H.

Proof. The proof is exactly the same as before.□

Proof of Theorem 2.1(b):

Lemma 2.3. *Let S be a finitely generated semigroup that is embeddable in a finitely generated semigroup H; suppose H has a representative function r in* F-DTime(T). *Then S has a representative function r_1 in H, with complexity* F-DTime($T(\mathrm{O}(.))$).

Proof. Let A be the set of generators of H and let $\{s_1, \ldots, s_m\} \subseteq A^+$ be representatives of a finite set of generators of S. Let us pick a new alphabet $B = \{b_1, \ldots, b_m\}$ of size m, disjoint from A; then $\langle A \cup B : R \cup \{b_1 \rightarrow s_1, \ldots, b_m \rightarrow s_m\}\rangle$ is a presentation of H (Tietze transformation). Let $\varphi : \mathrm{B}^+ \rightarrow \mathrm{A}^+$ be the homomorphism induced by the map $b_i \rightarrow s_i (i = 1, \ldots, m)$. This gives us a representative function $r_1 : \mathrm{B}^+ \rightarrow \mathrm{A}^+$ of S in H, defined by $r_1(x) = r(\varphi(x))$; clearly, r_1 belongs to F-DTime($T(\mathrm{O}(.))$) if the function r belongs to F-DTime(T).□

So, to prove Theorem 2.1(b) it is sufficient to prove that the reduction function $\mathsf{red}(.)$ of H belongs to F-DTime(W). As is often the case in similar situations (e.g. in the Proposition 3.3.b and Appendix A3), a complexity upper-bound of $\mathrm{O}(W(.)^2)$ would be much easier to prove than the bound $\mathrm{O}(W)$ that we will prove now.

When a rewrite system is simulated by a Turing machine, one problem is that rewrite

rules are usually not length-preserving, whereas the Turing machine tapes cannot stretch or shrink (except for insertion/deletion of one letter near the right endmarker). However, a simple way to build a Turing machine with "rubber tapes" is to replace each tape by two stacks (cut each tape at the position of the head and view each part of the tape as a stack). See Hopcroft and Ullman (1979) and Birget (1998).

Let $H = \langle A : R \rangle$ be a complete rational presentation with A finite. Let $\mathrm{M}_R = (\mathrm{Q}, A, \delta, \mathrm{q}_0, \mathrm{F})$ be a nondeterministic finite transducer which computes R; i.e. $(x, y) \in R$ iff on input x there is some accepting computation with output y. Here Q is the set of states, A is the input–output alphabet, q_0 is the start state, F is the set of accept states, and $\delta \subseteq \mathrm{Q} \times A^* \times \mathrm{Q} \times A^*$ is the transition relation. We will construct a deterministic Turing machine M which on input x computes $\mathsf{red}(x)$ in time $\leq W(|x|)$. An important part of M is a variant of the subset construction (see Hopcroft and Ullman (1979)) which simulates M_R deterministically in order to find subsegments of x that are left-sides of rules in R.

On input x the Turing machine M proceeds as follows:

M reads the input $x = x_1 \dots x_n$ (where each x_i is a letter $\in A$), from left to right and after reading any prefix $x_1 \dots x_i$ of x, M wants to know if any suffix of $x_1 \dots x_i$ is a left-side of a rule. For this purpose, M computes the set of states that M_R reaches when it reads this input. Moreover, at every step, M starts a new simulation of M_R; thus, after reading $x_1 \dots x_i$, M remembers a collection of sets of states of M_R, one set for each starting position of a simulation: after reading $x_1 \dots x_i$ the collection of sets remembered by M is

$$\{\{\mathrm{q}_0\}, \delta(\mathrm{q}_0, x_i), \delta(\mathrm{q}_0, x_{i-1}x_i), \dots, \delta(\mathrm{q}_0, x_2 \dots x_{i-1}x_i), \delta(\mathrm{q}_0, x_1x_2 \dots x_{i-1}x_i)\}.$$

Note that there are $\leq 2^{2^{|Q|}}$ different such collections (a finite number). At position i, the corresponding collection is written on the tape. As soon as an accept state ($\in$ F) appears in a set of the collection, M concludes that a suffix of $x_1 \dots x_i$ (the input read so far) is the left-side (let us call it α_1) of a rule of R. Now M marks this position and backtracks on the tape to find the beginning of α_1. It does this by using the accept state found, and reading $x_1 \dots x_i$ backwards while applying the transitions of M_R in reverse, until the start state of M_R is reached. Now the beginning of α_1 has been found.

Next, M simulates M_R to compute any output β_1 with input α_1; see the Proposition in Appendix A4 for a way to do this deterministically in linear time. In the process, α_1 is replaced by β_1 on the tape (we use rubber tapes, as explained earlier).

Now M places its head at the left end of β_1 and resumes the simulation of M_R and the computation of the collection of state sets at each position; the search for the next rule need not start left of β_1 because the state collections that were written on the tape to the left of β_1 are still correct. The next left-side of a rule $\alpha_2 \to \beta_2$ is found in the same way as for the first rule, and so on.

In summary, M repeatedly executes a loop consisting of: (1) a search phase for the right end of the left-side α_k of the next rule $\alpha_k \to \beta_k$, to be applied; and (2) a rewrite phase (in which the left end of α_k is found, and α_k is replaced by β_k; the head is then placed at the left end of β_k). M continues this way until no rules apply anymore. By completeness of the rewrite system R, the resulting word is now reduced, i.e. it is the word $\mathsf{red}(x)$. The Turing machine M is similar to the one in Section 2 of O'Dúnlaing (1983); however in O'Dúnlaing (1983) the hypotheses are more restrictive (the range of R is finite, which simplifies the construction), and the conclusions are much stronger than they could possibly be here.

Let us analyse the time-complexity of the Turing machine M. Let $(\alpha_k \to \beta_k : k = 1, \ldots, N)$ be the entire sequence of rules of R that the machine applies, starting with input x, to derive $\mathsf{red}(x)$. Let $\omega_k = |\alpha_k| + |\beta_k|$ be the work of the kth rule. The total time of the computation is the total time to search (for the positions of the right end of each α_k), plus the total time to find the left end of each α_k and to perform the rewriting for each rule $\alpha_k \to \beta_k$. The total time to find the left ends of all the α_k's is $\leq \sum_k |\alpha_k| \leq \sum_k \omega_k \leq W(|x|)$; and the total rewriting time is proportional to the rewrite work, thus it is $\mathrm{O}(\sum_k \omega_k) \leq \mathrm{O}(W(|x|))$.

Let us calculate the total search time for finding the position of the right end of each $\alpha_{k+1} (0 \leq k < N)$. To find the right end of α_{k+1} the machine will start at the left end of β_k, and read a string γ_{k+1} consisting of a prefix of β_k, or perhaps all of β_k, followed perhaps by portions of previously written β_i's $(i < k)$, followed perhaps by a portion of the input x.

We claim that every letter in the string γ_{k+1} is read once in the search phase, before it is rewritten (the tape positions of γ_{k+1} might be re-visited in a later search phase, but some rewriting will have happened at these positions before the next search): Indeed, if a letter of γ_{k+1} is to the left of α_{k+1}, it will not be visited again during a search phase, until it is rewritten (because the state set collection on such a letter is still valid); if a letter of γ_{k+1} is within α_{k+1} then it will be rewritten (as α_{k+1} is replaced by β_{k+1}).

Moreover, γ_{k+1} consists of substrings of some β_i's, $i \leq k$, (and of x) that are searched once and replaced by new β_i's, $i > k$, as γ_{k+1} is rewritten; thus, different γ_j's $(1 \leq j \leq N)$ will involve disjoint sets of portions of β_i's. Thus the total search time to find all the right ends of left-sides of the rules is thus $|\gamma_1| + \cdots + |\gamma_N| \leq |\beta_1| + \cdots + |\beta_N| + |x|$. This is bounded above by $\omega_1 + \cdots + \omega_N + |x| \leq 2W(|x|)$. This complexity analysis has some similarity with the proof of Theorem 2.2.9 of Book and Otto (1993); there R is finite, however, which makes a great difference.□

3. Infinite Presentations of Monoids and Groups

In this section it is shown, roughly speaking, that every finitely generated monoid or group with recursively enumerable word problem has an (infinite) presentation $\langle A : R \rangle$ where R has low complexity, and in which the work distance is only polynomially bigger than the complexity of the best algorithm (deterministic or nondeterministic). In this section we consider only *monoids* (and groups).

The first proposition gives a finer version of the trivial fact that a finitely generated monoid or group has a recursively enumerable word problem iff it has a presentation $\langle A : R \rangle$ where A is finite and R is recursively enumerable.

Proposition 3.1. (Recursively enumerable word problems) *A finitely generated monoid S has a recursively enumerable word problem iff S has a presentation $\langle A : R \rangle$ (which is a group presentation if S is a group) with A finite, such that the set of relations R is the intersection of two deterministic context-free languages and R belongs to the complexity class $\mathcal{AC}^0$.*

Since every DCFL is in DTime(linear) (see Hopcroft and Ullman (1979)), R also belongs to DTime(linear).

The second proposition is a refinement of the known (and nontrivial) fact that a finitely generated monoid has a decidable word problem iff it has a presentation $\langle A : R \rangle$

where A is finite, dom$R$ is decidable, and R is decidable and complete (i.e. confluent and terminating). By "dom$R$" we mean the set of all left-sides of R (the *domain* of R).

PROPOSITION 3.2. (DECIDABLE WORD PROBLEMS) *A finitely generated monoid S has a decidable word problem iff S has a* complete *presentation $\langle A : R\rangle$ where A is finite, the domain of R is recursive, and R is the intersection of two deterministic context-free languages and belongs to the complexity class* I/O-DTIME(linear) $\cap\mathcal{AC}^0$.

The above proposition and the next one improve the previously mentioned result of Madlener and Otto (1988, Theorems 3.5 and 4.2) about infinite context-sensitive presentations and the Grzegorczyk hierarchy ; it also gives a converse (which was not possible in the formulation of Madlener and Otto (1988)). We still assume that time complexity bounds are positive and superadditive (and total recursive).

PROPOSITION 3.3. (NONDETERMINISTIC TIME-COMPLEXITY OF WORD PROBLEMS) *(a) Let S be any finitely generated monoid or group. If the word problem of S is in* NTIME(T) *then S has a presentation $\langle A : R\rangle$ (which is a group presentation if S is a group) with A finite, such that R is the intersection of two deterministic context-free languages and belongs to the complexity class* DTIME(linear) $\cap\mathcal{AC}^0$*; moreover the work distance of $\langle A : R\rangle$ is bounded above by* $\mathrm{O}(T(.)^2)$ *(i.e. $\omega(x,y) \le c\cdot T(|x|+|y|)^2$ for all words x, y with $x \underset{S}{=} y$, where $c > 1$ is a constant).*

(b) Conversely, let $\langle A : R\rangle$ be a semigroup presentation of a monoid S with A finite and R in DTIME(linear)*, and such that the work distance is bounded above by a total recursive function W. Then the word problem of S is in* NTIME(W).

Proposition 3.3 immediately yields another characterization of the nondeterministic time-complexity of the word problem of monoids; compare this with the characterization in Birget (1998). Recall that $T(.)^{\mathrm{O}(1)}$ denotes the set of all functions f for which there exists a constant $c > 0$ such that $f(n) \le T(n)^c$ (for all n).

COROLLARY 3.4. *A finitely generated monoid S has its word problem in* NTIME$(T(.)^{\mathrm{O}(1)})$ *iff S has a presentation $\langle A : R\rangle$ (which is a group presentation if S is a group) with A finite, such that:*

- *the set of relations R is the intersection of two deterministic context-free languages, and*
- *the work distance of $\langle A : R\rangle$ is bounded above by some function in* $\mathrm{O}(T(.)^{\mathrm{O}(1)})$.

We can, in particular, characterize the class of word problems in NP:

A finitely generated monoid S has its word problem in NP iff S has a presentation $\langle A : R\rangle$ (which is a group presentation if S is a group) such that A is finite, R is the intersection of two deterministic context-free languages, and the work distance of $\langle A : R\rangle$ is bounded above by a polynomial.

For deterministic complexity we obtain a result that is weaker and more complicated. Due to the close connections with the word problem and nondeterminism, it is more difficult to find tight general connections between the word problem and deterministic time complexity.

PROPOSITION 3.5. (DETERMINISTIC TIME-COMPLEXITY OF WORD PROBLEMS) *(a) Let S be a finitely generated monoid, given by a congruence $\equiv_1$ on A_1^+, where A_1 is finite. Assume $\equiv_1$ has a representative function r_1 in* F-DTIME(T). *Then S has a monoid presentation $\langle A : R\rangle$ with A finite, and with the following properties:*

(1) R is confluent;

(2) for every $z \in A^+$ there is a left-most greedy derivation $z \xrightarrow{*} r(z)$ with work $O(T(|z|)^2)$;

(3) the congruence of $\langle A : R\rangle$ on A^+ has a representative function r which belongs to F-DTIME(T); *we have $A_1 \subseteq A$ and r agrees with r_1 on A_1^+;*

(4) the set of relations R is the intersection of two deterministic context-free languages and belongs to the complexity class $\mathcal{AC}^0$;

(5) R is a total function (so the set of left-sides of rules in R is A^+) and belongs to I/O-DTIME(linear).
If a reduction order $\geq_1$ is defined on A_1^+ with respect to which r_1 is the $\geq_1$-minimum representative function, then S has a monoid presentation $\langle A : R'\rangle$ with A finite, and where R' is complete; *the above properties (1)–(4) still hold, but (5) is replaced by*

(5') R' is a partial function belonging to I/O-DTIME(T); *in particular, the set of left-sides of R' belongs to* DTIME(T).

(b) Conversely, let $\langle A : R\rangle$ be a presentation where A is finite and R is a partial function which belongs to I/O-DTIME(linear); *assume that $\langle A : R\rangle$ has a representative function r such that for every $x \in A^+$, there is a left-most greedy derivation $x \xrightarrow{*} r(x)$, and the work of this derivation is $\leq W(|x|)$, where $W(.)$ is total recursive. Then the representative function r belongs to* F-DTIME($O(W(.)^3)$).

The proofs of the "left-to-right" implications in the four propositions are based on *Craig's trick* (see Cohen (1989, p. 256), Hodges (1993, p. 269), and Craig (1953)).

PROOF OF PROPOSITION 3.1. The right-to-left implication is straightforward. If the set of relations R is in DTIME(linear), or much more generally, if R is recursively enumerable, then the word problem is also recursively enumerable.

Let us prove the harder left-to-right implication. Let $S = \langle A_1 : R_1\rangle$ be any monoid or group presentation, where A_1 is finite, and where R_1 is recursively enumerable and accepted by a one-tape Turing machine M; it will not matter here whether M is deterministic or nondeterministic. If the above is a group presentation, R_1 is a subset of A_1^*; if it is a monoid presentation, R_1 is a subset of $A_1^* \times A_1^*$. Let Γ be the alphabet of M (so $A_1 \subseteq \Gamma$) and let Q be the state set of M; let ¢ and \$ be the left, respectively right, endmarkers on the tapes. A configuration of the machine M is a word in ¢Γ^*QΓ^*\$ (see Appendix A1 for details about Turing machines). The input belongs either to A_1^* (for groups), or to $A_1^* \times A_1^*$ (for monoids); in the latter case we represent the input $(x, y) \in A_1^* \times A_1^*$ by the string $x\%y \in (\{\%\} \cup A_1)^*$, where % is a new letter ($\notin A_1$). The initial configuration on input z ($z \in A_1^*$, or $z = x\%y$, where $(x, y) \in A_1^* \times A_1^*$) has the form $C_0(z) = ¢q_0z\$$. An accepting computation of M on input z is a sequence of configurations of the form $C_0(z), C_1, \ldots, C_i, C_{i+1}, \ldots, C_{t-1}, C_t$, where C_t is an accepting configuration, and C_{i+1} follows from C_i by application of one transition of M (for all i, $0 \leq i < t$); we can assume without loss of generality that all accepting computations have even length (i.e. t

is even). We consider the following language:

$$\mathrm{L}_0 = \{\mathrm{C}_0(z)^{\mathrm{rev}}\#\mathrm{C}_1\#\mathrm{C}_2^{\mathrm{rev}}\#\ldots\#\mathrm{C}_{2i-1}\#\mathrm{C}_{2i}^{\mathrm{rev}}\#\ldots\#\mathrm{C}_t^{\mathrm{rev}} : z \in A_1^*\%A_1^*, (\text{or } \in A_1^*),$$
$$\text{and } \mathrm{C}_0(z), \mathrm{C}_1, \ldots, \mathrm{C}_{t-1}, \mathrm{C}_t \text{ is an accepting computation of M on input } z\}.$$

NOTATION $(.)^{\mathrm{rev}}$ indicates reversal of a string; # is a new letter. The configurations in L_0 are written alternatingly in reverse; we shall see that the purpose of this is to make L_0 the intersection of two DCFLs.

The alphabet of L_0 is $A_2 = \mathrm{Q} \cup \Gamma \cup \{¢, \$, \#, \%\}$.

Next, we consider the map $\theta : A_2^* \to (\Gamma \cup \{\%\})^*$ which, first, sends any word $u_1\#u_2 \ldots u_{m-1}\#u_m$ (where u_i does not contain #, $1 \le i \le m$) to u_1^{rev} and then erases all occurrences of ¢, \$, and elements of Q. In particular, θ sends an accepting computation on input z to this input z; thus $\theta(\mathrm{L}_0) = R_1$.

We will use a new copy A_2'' of the alphabet A_2; A_2'' is in one-to-one correspondence with A_2 and disjoint from A_2. For a word $w \in A_2^*$, the corresponding copy of w over the new alphabet A_2'' is denoted by w'' (obtained from w by replacing each letter in A_2 by the corresponding letter in A_2''). Now we give a new presentation of S which has low complexity.

We consider the monoid presentation $\langle A : R\rangle$ where:

$$A = A_2'' \cup A_1, \text{ and}$$
$$R = \{(x, yw'') : w \in \mathrm{L}_0, \theta(w) = x\%y, (x, y) \in R_1\} \cup \{(a'', 1) : a'' \in A_2''\}.$$

Recall that $\theta(\mathrm{L}_0) = R_1$. Here 1 is the empty word.

If S is a *group*, we consider the group presentation $\langle A : R\rangle$ where:

$$A = A_2'' \cup A_1, \text{ and}$$
$$R = \{\theta(w).w'' : w \in \mathrm{L}_0\} \cup \{a'' : a'' \in A_2''\}.$$

Clearly, the new presentation $\langle A : R\rangle$ is a presentation of the original monoid or group S.

Let us analyse the complexity of R. The sets $\{(a'', 1) : a'' \in A_2''\}$ and $\{a'' : a'' \in A_2''\}$ are finite, so they do not affect the complexity.

CLAIM. The sets $\mathrm{L}_0, \{\theta(w).w'' : w \in \mathrm{L}_0\}$, and $\{(x, yw'') : w \in \mathrm{L}_0, \theta(w) = x\%y, (x, y) \in R_1\}$ belong to DTIME(linear) $\cap\, \mathcal{AC}^0$. Moreover, the languages $\mathrm{L}_0, \{\theta(w) \cdot w'' : w \in \mathrm{L}_0\}$, and $\{x\%yw'' : w \in \mathrm{L}_0, \theta(w) = x\%y, (x, y) \in R_1\}$ are each the intersection of two DCFLs. (The third language is just R, in the monoid case, with (x, y) always replaced by $x\%y$ in order to turn R into a set of strings.)

PROOF OF CLAIM. It is straightforward to give a linear-time deterministic algorithm for checking that a string is a well-formed accepting computation of a Turing machine; the algorithm just has to make sure that successive configurations are equal except for the two letters surrounding the state symbol (where a transition is applied). See Section 8.6 in Hopcroft and Ullman (1979) for more details.

Regarding the $\mathcal{AC}^0$-property of L_0: Equality of two strings (except for three symbols near the position of the state symbol) can be checked by a constant-depth acyclic boolean circuit with a linear number of gates; reversal of string is also easy to do in $\mathcal{AC}^0$. (See also Birget (1996) where a very similar case is considered, and see van Leeuwen (1990) for more on $\mathcal{AC}^0$.)

Section 8.6 in Hopcroft and Ullman (1979) shows that L_0 is the intersection of two

deterministic context-free languages. The same proof also applies to $\{\theta(w)\cdot w'' : w \in \mathrm{L}_0\}$ and to $\{x\%yw'' : w \in \mathrm{L}_0, \theta(w) = x\%y, (x,y) \in R_1\}$.□

Proof of Proposition 3.2. Here again, the right-to-left implication is easy. If a presentation is complete and if the set of relations and the set of left sides are decidable then the word problem is decidable too.

For the main (left-to-right) implication, let $S = \langle A_1 : R_2\rangle$ be any presentation of the monoid S with A_1 finite. Let $\geq_{\mathrm{llex}}$ be the length-lexicographic order on A_1^* (words are ordered first by length, then equally long words are ordered lexicographically); this is a reduction order (as defined in the Introduction). For each $x \in A_1^*$ let $\underline{x}$ be the minimum element (with respect to $\geq_{\mathrm{llex}}$) in the congruence class of x.

When S has a decidable word problem, the function $u \in A_1^* \to \underline{u} \in A_1^*$ is total recursive. Therefore the set $R_1 = \{(x,\underline{x}) : x \in A_1^*, x \neq \underline{x}$, and every strict subsegment of x is reduced$\}$, is decidable. Clearly, $\langle A_1 : R_1\rangle$ is a presentation of S which is confluent and terminating (it is obtained from $\{(x,\underline{x}) : x \in A_1^*\}$ by removing all "redundant rules", see Sims (1994) and Epstein *et al.* (1992)).

Now we apply the construction of Proposition 3.1 to the semigroup presentation $\langle A_1 : R_1\rangle$. We obtain the monoid presentation $\langle A : R\rangle$ of S where: $A = A_2'' \cup A_1$, and $R = \{(x,\underline{x}w'') : w \in \mathrm{L}_0, \theta(w) = x\%\underline{x}, (x,\underline{x}) \in R_1\} \cup \{(a'',1) : a'' \in A_2''\}$. Here L_0 is obtained from a deterministic Turing machine M which, on input x, computes $\underline{x}$ and then checks whether $(x,\underline{x}) \in R_1$.

By the proof of Proposition 3.1, R is the intersection of two deterministic context-free languages and R is in $\mathcal{AC}^0$. It is straightforward to check that the rewriting system R is terminating and confluent. Moreover, since the Turing machine M for L_0 is deterministic, it is straightforward to check that R belongs also to I/O-DTime(linear).□

Proof of Proposition 3.3(a). Let S be a finitely generated monoid or group with word problem in NTime(T), with respect to a finite set of generators A_1 and a congruence $\equiv$. In the monoid case, we let R_1 be the entire congruence relation $\equiv$. If S is a group we let R_1 be the congruence class of the empty word 1. Let M be a nondeterministic Turing machine which decides the word problem of S in time $\leq T(.)$. We apply the construction given in the proof of Proposition 3.1 to this presentation $\langle A_1 : R_1\rangle$ and to M. The construction will have to be slightly modified because M is a multi-tape Turing machine (with k tapes): we represent a configuration of the multi-tape Turing machine by one word as in the k-to-one tape conversion (Appendix A2). Next, the language L_0 is defined as in the proof of Proposition 3.1: L_0 consists of words of the form $\mathrm{C}_0(z)^{\mathrm{rev}}\#\mathrm{C}_1\#\mathrm{C}_2^{\mathrm{rev}}\#\ldots\#\mathrm{C}_t^{\mathrm{rev}}$, where each C_i is the one-tape representation of a configuration of the multi-tape machine M and C_{i+1} follows from C_i by application of one transition of M (for all $i, 0 \leq i < t$). (Note we only use the conversion as a way to represent multi-tape configurations by single words; we keep the transitions of the multi-tape machine M. Recall that a similar procedure was used in the proof of Theorem 2.1(a), the "case of rational presentations".)

As in the proof of Proposition 3.1 we then obtain a presentation $\langle A : R\rangle$ with A finite, such that R is the intersection of two deterministic context-free languages and belongs to the complexity class $\mathcal{AC}^0$. Moreover, we have:

Claim. The work distance in $\langle A : R\rangle$ is $\leq \mathrm{O}(T(.)^2)$.

Proof. Let $x, y \in A^*$ be two words that are equivalent with respect to $\langle A : R\rangle$. We

prove that the work distance $\omega(x, y)$ is $\leq c \cdot T(|x| + |y|)^2$ for monoid presentations; for groups the proof is essentially the same.

Case 1: x and y contain letters of A_1 only. Then $x\%y$ is accepted by the k-tape Turing machine M, and there is an accepting computation $C_0(x\%y), \mathrm{C}_1, \ldots, \mathrm{C}_{t-1}, \mathrm{C}_t$ of duration $t \leq T(|x|+|y|)$. Note that every configuration (represented as a word, via the k-to-1 tape conversion) has length $|\mathrm{C}_i| \leq t + 3$, since the accepting configuration C_t has length 3 and since in time t the length of a tape can change by $\leq t$ (the "3" comes from the two endmarkers and the state symbol). So the word $w = \mathrm{C}_0(x\%y)^{\mathrm{rev}}\#\mathrm{C}_1\#\mathrm{C}_2^{\mathrm{rev}}\#\ldots\#\mathrm{C}_t^{\mathrm{rev}}$ has length $|w| \leq (t+3)^2 \leq c \cdot T(|x| + |y|)^2$, for some constant c. We now derive y from x as follows: First, from x we go to yw'' (where w'' is the copy of w over the alphabet A_2''), using the relation (x, yw''); this is one step, but the corresponding work is $|x| + |y| + |w| \leq (c+1) \cdot T(|x| + |y|)^2$ (by the fact that $T(n) > n$ for all n). Next, we erase w'' using relations $\{(a'', 1) : a'' \in A_2''\}$, and this involves $|w| \leq c \cdot T(|x|+|y|)^2$ steps, each with work 1. So the total work to derive y from x is $\leq \mathrm{O}(T(|x| + |y|)^2)$.

Case 2: x and y are arbitrary equivalent words over the alphabet A. This case is first reduced to Case 1: From x and y we derive words x_1 and y_1, with letters in A_1 only; this uses $\leq |x| + |y|$ erasing steps of the form $a'' \to 1\,(a'' \in A_2'')$, and each step has work 1. Next, as in Case 1, x_1 derives y_1 using work $\leq \mathrm{O}(T(|x_1| + |y_1|)^2) \leq \mathrm{O}(T(|x| + |y|)^2)$. Therefore, $\omega(x, y) \leq |x| + |y| + \mathrm{O}(T(|x| + |y|)^2) \leq \mathrm{O}(T(|x| + |y|)^2)$ (the latter inequality holds because $T(n) > n$ for all n).□

Proof of Proposition 3.3(b): Here we assume that our monoids are given by semigroup presentations (and the groups by group presentations); it is well known that a monoid can be given a semigroup presentation (by adding a letter for the identity), and this changes the work by a constant multiple at most.

Claim. Assume that the monoid S has a semigroup or group presentation $\langle A : R\rangle$ such that A is finite, R belongs to DTime(linear), and the work distance ω of $\langle A : R\rangle$ is bounded above by $W(.)$. Then the word problem of S belongs to NTime(W).

Proof. The proof of the claim is very similar to the proof of Proposition 3.4 of Birget (1998) (the Converse of the Isoperimetric Embedding Theorem). To make this paper self-contained, we include the proof. We only consider the case of a semigroup presentation; the group case is similar (and easier). The proof is based on van Kampen diagrams. The following notion, inspired directly from scheduling theory, will be useful:

The *precedence graph* Π associated with a semigroup diagram K with boundary label (x, y), is a directed acyclic atransitive graph (i.e. the Hasse diagram of a partial order). The vertices of Π are the K-cells, i.e. the bounded faces of the semigroup diagram K. The Π-edges incident to a K-cell C with boundary label (u, v) are defined as follows: If the cell is used in the direction $u \to v$ (by Fact 3.1 of Birget (1998), this is uniquely determined by K), then there is a Π-edge from C to each cell that has a K-edge in common with the v-side of C. Similarly, there is a Π-edge into C from each cell that has a K-edge in common with the u-side of C. The roots of Π are the K-cells whose input side is entirely contained in the input side (labeled by x) of K.

The precedence graph tells us the possible orders in which rules can be applied in any derivation that corresponds to a given semigroup diagram.

(a) *Turing machine construction:* From $\langle A : R\rangle$ we construct a nondeterministic Turing

machine M_R with two rubber tapes, which on input $x\#y$ (where $x, y \in A^+$ and $\#$ is a new symbol, not in A) derives a word y_1 from x, by repeatedly guessing relations $\in R$ and positions where the relations are applied. The guessing is done nondeterministically; only one rubber tape is used for this. Then $x \underset{S}{=} y$ iff there exists such a nondeterministic computation which turns x into y; thus, in this case the computation produces the word $y\#y$. At the end of the computation the machine verifies that the output $y_1\#y$ satisfies $y_1 = y$; the two tapes are used in order to do this verification in linear time. Obviously, this machine correctly decides the word problem (nondeterministically) since it simulates all possible derivations and cannot do anything else to a word.

We want to prove that if $x \underset{S}{=} y$ and $\omega(x, y) \leq \mathrm{O}(W(n))$ with $n = |x| + |y|$, then the Turing machine M_R has at least one accepting computation on input $x\#y$ with time complexity $\leq c.W(n)$. Here $c > 0$ is a constant.

(b) *A bound* $\mathrm{O}(W(.)^2)$: Let us first prove that M_R decides the word problem in time $\leq c.W(n)^2$. After that we will prove a sharper upper bound $\mathrm{O}(W(.))$.

To carry out the rewriting itself the machine needs time $\leq \omega(x, y)$. In addition, time will be spent to check whether a guessed relation (α, β) is indeed in R; this takes time $\leq c_1 \cdot (|a| + |b|)$ for some constant c_1, since R belongs to DTime(linear). Moreover, the machine has to move its head to the place in the word (derived so far) where the next relation is to be applied. The word derived at any moment and currently stored on the tape, has length $\leq \omega(x, y)$; the machine never needs to move farther than this on the tape to find the place where it will apply the next rule in the derivation, so the time between successive rewrites is $\leq \omega(x, y)$. This gives a total time $\leq \mathrm{O}(\omega(x, y)^2)$, to which one has to add time $\leq \mathrm{O}(|y|)$ for checking at the end of the computation (using two tapes) that $y_1 = y$. Since $W(n) \geq n$, we have $|y| < W(|x| + |y|)$. This gives an upper bound of $\mathrm{O}(W(.)^2)$ for the time complexity of M_R.

We will now show that M_R also has accepting computations of duration $\mathrm{O}(W(.))$. Such computations will nondeterministically simulate certain derivations that minimize the movement along the tape while moving to the next position where a rule should be applied.

(c) *Proof of the bound* $\mathrm{O}(W(.))$: The proof of this tight bound uses semigroup van Kampen diagrams, and is based on the following simple fact: applying a rule $u \to v$ to a word that contains u is equivalent to removing a cell labeled by (u, v) from the diagram whose input boundary contains u. See Figure 2.

The general shape of a semigroup diagram is a "path of balls", as illustrated in Figure 1; recall that there can only be one source and one sink (see Higgins (1992), Remmers (1980) and Birget (1998)). Let (x, y) be the boundary label of a semigroup diagram K. By Proposition 1.1, K has $\leq 2W(n)$ edges, where $n = |x| + |y|$. We nondeterministically select the following computation of our Turing machine M_R. In this computation M_R reads the input x from left to right until it arrives at a ball of the diagram; the time to do this is a fraction of the length $|x|$, and this portion of the input will never be read again (except in the very end, when the machine checks, in linear time, that $y_1 = y$). (Of course, M_R does not "know" that it is at a ball; it just guesses this.) Now the machine guesses a cell C of the ball, such that C has at least one edge on the outer boundary of K at the current position of the head. If the cell C is a root of the precedence graph Π of K, then C can be removed by applying the relation $u \to v$ which labels C (if C is a root of Π then the path labeled by u belongs to the input boundary of K, labeled by

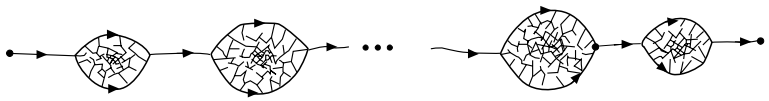

Figure 1. A typical semigroup diagram.

Figure 2. Removal of a cell by application of a rule $u \to v$.

x); see Figure 2. Since R belongs to DTime(linear), the cell C can be guessed in time $O(|u| + |v|)$.

If the currently visited cell C is not a root of the precedence graph Π, the machine carries out a nondeterministic *depth-first search* in Π, starting at C in the back-track phase. More precisely, in the back-track phase M_R moves right on the tape until it finds a root; it removes this root cell (by applying the corresponding relation, as on Figure 2). Then it moves left again (forward phase of the depth-first search) as long as the cells it sees along the outer boundary are roots of the diagram obtained thus far, and applies the corresponding relations (thus removing these root cells). When the currently visited cell is not a root (and if the cell C, from which the back-track search started, has not been reached yet), the machine back-tracks again (by moving right until it finds a root of the precedence graph Π). Eventually, after several back-tracks and forward searches, the cell C from which this search started will become a root (once all its predecessors in the precedence graph are removed), and then C is removed.

The total time it took to remove C (from the start of the back-track at C, until the removal of C), is proportional to the total work of the cells removed so far: Indeed all the cells that touch the outer boundary and that are on the way from C to the left-most root (that are predecessors of C in Π) are removed in the process; so the movement of the head, from the position of C to the position of this root, is bounded by the work of the cells removed. Similarly, in any other back-track (from a cell to a root), the cells visited on the way are eventually removed during that part of the search. Therefore, the total time to remove the ball (thus replacing it by its output boundary, labeled by a subsegment of y) is big-O of the number of edges of the removed ball.

When a ball has been removed the machine reads the next input portion until it reaches the next ball, and starts removing this ball. Overall, the selected computation takes time big-O of the number of edges of the diagram, and this is $O(W(.))$.□

Proof of Proposition 3.5 (a). Let M be a deterministic (multi-tape) Turing machine with time-complexity T, which on input x computes the representative $r_1(x)$. As in the proof of Proposition 3.1 and the other propositions, we define the language L_0:

$$L_0 = \{C_0(x)^{\mathrm{rev}} \# C_1 \# C_2^{\mathrm{rev}} \# C_3 \# \ldots \# C_t^{\mathrm{rev}} : x \in A_1^+, \text{ and } C_0(x), C_1, C_2, \ldots, C_t \text{ is the computation of M on input } x, \text{ producing the output } r_1(x)\}.$$

The new *presentation* of the monoid S is defined as follows: We pick $A = A_1 \cup A_2''$, where A_2'' is exactly as in the proof of 3.1. The set of relations R is now defined as follows (where the map θ was defined in the proof of 3.1):

$$\begin{aligned} R = \{(z, w''r_1(x)) : z \in A^+, \\ & x \in A_1^+ \text{ is obtained from } z \text{ by erasing all letters in } A_2'', \\ & w \in \mathrm{L}_0 \text{ and } x = \theta(w)\} \\ \cup \{(a'', 1) : a'' \in A_2''\}. \end{aligned}$$

Complexity of R: As in the previous proofs, one checks that R is the intersection of two deterministic context-free languages (when we write R as $\{z\%w''r_1(x) : \ldots\}$); this is based on the alternation between configurations and reverses of configurations. R also belongs to $\mathcal{AC}^0$ since palindromes are in $\mathcal{AC}^0$. Since M is deterministic, R is a function; moreover, R is a total function. It is straightforward to check that R belongs to I/O-DTime(linear): by the determinism of M, one can deterministically generate $w''r_1(x)$ on input z ; this takes time $\mathrm{O}(|w''r_1(x)|)$ since $w''r_1(x)$ is just a description of the computation of M.

The *representative function* r of the presentation $\langle A : R\rangle$ is defined as follows: $r(z)$ is obtained by first erasing all letters in A_2'' of z (call the new word $\epsilon(z)$), and then applying r_1 to $\epsilon(z)$. This is indeed a representative function: First, any word $z \in A^+$ is equivalent to $r(z)$, since $r(z)$ is obtained from z by applying rules of R. Second, if z and z' are equivalent then $r(z) = r(z')$; this can be proved by induction on the length of a derivation from z to z': When $z_i \to z_{i+1}$ via a rule $(a'', 1)$, then $r(z_i) = r(z_{i+1})$; and when $z_i = us_iv \to uw''r(s_i)v = z_{i+1}$ via a rule $(s_i, w''r_1\epsilon(s_i))$, then $r(z_{i+1}) = r(uw''r_1\epsilon(s_i)v) = r_1\epsilon(uw''r_1\epsilon(s_i)v) = r_1\epsilon(ur_1\epsilon(s_i)v)$, since w'' is over A_2''; and the latter is equal to $r_1(\epsilon(u)r_1\epsilon(s_i)\epsilon(v))$, since $r_1\epsilon(s_i)$ contains no letter of A_2''; this now equals $r_1(\epsilon(u)\epsilon(s_i)\epsilon(v))$ since r_1 is a representative function; the latter equals $r_1\epsilon(us_iv) = r(z_i)$.

Let us look at *left-most greedy derivations*. Let z be a word in A^+ such that $x \neq r(x)$, where x is obtained from z by erasing all letters in A_2''. We have a left-most greedy derivation $z \to w''r_1(x) \xrightarrow{*} r_1(x)$, where the second part of the derivation consists of erasing w''; the work of this derivation is $|z| + |r_1(x)| + 2|w''| = \mathrm{O}(T(|z|)^2)$.

Confluence: Any derivation δ starting from a word $z \in A^+$ leads to some word z_δ which is equivalent to z in $\langle A : R\rangle$; thus $r(z) = r(z_\delta)$. To z_δ we can then apply the left-most greedy derivation $z_\delta \xrightarrow{*} r(z_\delta)(= r(z))$; the result is $r(z)$, for every δ.

Termination: if r_1 is based on a reduction order $\leq_1$ on A_1^*: The monoid S is generated by $A = A_1 \cup A_2''$ as above, but the set of relations R' is now defined as follows:

$$\begin{aligned} R' = \{(z, w''r_1(x)) : z \in A^+, x \neq r_1(x), \\ & x \in A_1^+ \text{ is obtained from } z \text{ by erasing all letters in } A_2'', \\ & w \in \mathrm{L}_0 \text{ and } x = \theta(w)\} \\ \cup \{(a'', 1) : a'' \in A_2''\}. \end{aligned}$$

So, the only change is that now we require $x \neq r_1(x)$ in the presentation; all the previously proved properties can be proved again. Regarding the complexity of R', the property $x \neq r_1(x)$ can be checked by a deterministic pushdown automaton, since $r_1(x)$ appears reversed in w; also, palindromes (and nonpalindromes) can be recognized by $\mathcal{AC}^0$-circuits. However, R' is not a total function; the domain of R' (i.e. the left-sides of all the rules) is $\{z : x \neq r_1(x)$ where x is obtained from z by erasing all letters of $A_2''\}$.

Since r_1 belongs to F-DTIME(T), the domain of R' belongs to DTIME(T). Note that now R' is in I/O-DTIME(T) (instead of I/O-DTIME(linear)), because when x is not in the domain of R' it takes time $O(T(|x|))$ on input x to decide that there is no output.

We extend $\leq_1$ to a reduction quasi-order on A^*, by defining $z \leq z'$ iff [either $x <_1 x'$, or $x = x'$ and $|z| \leq |z'|$]; here x and x' are obtained from z, respectively z', by erasing all the letters in A_2''. This defines a quasi-order (reflexive and transitive, but not antisymmetric); it is clearly compatible with concatenation; moreover there are no infinite descending chains. Every rule of R causes a strict decrease in this reduction quasi-order (here we use the fact that $x \neq r_1(x)$ in the rules); thus, the rewrite system is terminating.□

PROOF OF PROPOSITION 3.5(B). We consider a deterministic Turing machine which, in a straightforward way, simulates the left-most greedy derivation on a string x, that produces $r(x)$. This derivation is unique since R is a function.

In order to apply a rule to the string z derived so far, the machine must find the left-most and longest subsegment of z which is the left-side of a rule. The machine does this as follows: it has two pointers (which are just new letters) on the tape which contains z; initially the pointers are at the ends of z. The machine checks whether the subsegment of z between the pointers is a left-side of a rule; if it is not, the right pointer is moved left one step, and the new subsegment is checked. This goes on until a left-side of a rule is found, or the right pointer reaches the left pointer. In the latter case, the left pointer is moved one step to the right, and the right pointer goes back all the way to the right end of z. Now the procedure starts over. This way, all subsegments are examined in the left-first greedy order, until a left-side of a rule is found.

Looking at all possible subsegments of z takes time $O(|z|^2) = O(W(|x|)^2)$ (the last relation holds because $|z| \leq W(|x|)+|x|$, since z was derived from x). For each subsegment of z considered, we run the machine for R in order to produce the corresponding right-side of a rule (if no right side exists, we will look at other subsegments). By the time complexity assumption on R it takes time $O(|u| + |v|)$ to compute v from u when (u, v) is a rule; and when there is no rule with left-side u it takes time $O(|u|)$ to find this out. Thus, the total time it takes to check subsegments of z that are not left-sides of rules, is $O(|z|^2) = O(W(|x|)^2)$. Thus (letting d stand for the length of the derivation, and letting (u_i, v_i) denote the rule applied at the ith step in the derivation), the total time of the simulation will be: $\sum_{1\leq i\leq d}(O(W(|x|)^2) + O(|u_i| + |v_i|)) = O(d \cdot W(|x|)^2) + O(W(|x|)) = O(W(|x|)^3)$ (the latter holds since $d \leq W(|x|)$).

4. Discussion

The embedding in the results of Section 2 is necessary in general. Indeed, Kobayashi (1995) proved recently that *there exist finitely presented monoids in which the word problem is decidable, but that have no complete presentation in which the set of left sides of the rules is regular.*

4.1. OPEN PROBLEMS

A problem from Bauer (1981) (see also Otto (1989)):

Can every finitely generated (semi)group with decidable word problem be embedded into a (semi)group with a finite complete *presentation ?*

The most likely answer (to me) is that this is not true. If it were true, there would

also be the question of whether the relation between derivation work and deterministic complexity could be polynomial (as in Theorem 2.1 and Corollary 2.3). (However for term rewriting there are results in the positive direction, see Meseguer and Goguen (1985) and Bergstra and Tucker (1979).)

It is clear that if a congruence on A^+ (with A finite) has a representative function in F-DTIME(T) then the word problem is in DTIME(T). What about the converse? A natural, negative conjecture is:

There are finitely presented semigroups in which the word problem is in P (deterministic polynomial time), but that have no representative function belonging to F-DTIME(n^k), *for any k.*

Intuitive motivation for the conjecture: Compare this with the optimization problems (e.g. the Traveling Salesman Problem) where the problem of finding an optimal solution is conjectured to be of "higher complexity" than the corresponding decision problem.

Can the results from Section 3 be improved so that R belongs to **F**-DTIME(linear), or at least **F**-DTIME(polynomial), while keeping the derivation work polynomially bounded by T? In Section 3 we only have I/O-DTIME(linear).

Appendix

A1. TURING MACHINES, NOTATION AND DEFINITIONS

In this paper we use the same Turing machine model as in the standard textbook (Hopcroft and Ullman, 1979), except for the treatment of the infinite blank part of the tapes. We often use k-tape Turing machines ($k \geq 1$), but we only need detailed notation for one-tape machines. A one-tape Turing machine is a structure $\mathrm{M} = (\mathrm{Q}, \Sigma, \Gamma, ¢, \$, \delta, \mathrm{q}_0, \mathrm{F})$, where Q is the set of states, Σ is the input alphabet, Γ is the total alphabet (such that $\Sigma \subseteq \Gamma$); ¢ and \$ is the left, respectively right, endmarker symbol (¢ and \$ do not belong to Γ); $\mathrm{q}_0 (\in \mathrm{Q})$ is the start state, $\mathrm{F} (\subseteq \mathrm{Q})$ is the set of accept states; all these sets are finite. Finally, δ is a set of transitions of the form $(\mathrm{q}, \mathrm{b}) \rightarrow (\mathrm{q}', \mathrm{c}, \epsilon)$, where $\mathrm{q}, \mathrm{q}' \in \mathrm{Q}, \mathrm{b}, \mathrm{c} \in \Gamma \cup \{¢, \$\} \cup \Gamma\$, \epsilon \in \{-1, +1\}$. The meaning of a transition will be explained shortly.

Every tape is divided into cells, each of which holds one letter of Γ (except for the leftmost cell, which holds ¢, and the rightmost cell, which holds \$). Note that in this version of a Turing machine all tapes are finite at any moment; there is no "blank symbol". (In connection with the word problem, the usual view of the Turing machine tape as infinite but mostly blank, is not convenient.)

The head points to a cell. A *configuration* is of the form ¢upv\$, where p is the current state, $¢\mathrm{uv}\$ \in ¢\Gamma^*\$$ is the current content of the tape, and the position of p between u and v indicates that the head points to the left-most letter of v (or to \$ if v is empty, or to ¢). For this notation to be unambiguous we must assume that Q and Γ are disjoint. An initial configuration on input $\mathrm{w} \in \Sigma^*$ is of the form $¢\mathrm{q}_0\mathrm{w}\$$. We can (and will) assume that configurations of the form p¢v\$ never occur: this can be done by adding new letters to the total alphabet Γ and by adding transitions on these new letters that make sure that ¢ is never reached.

Types of transitions $(q, b) \to (q', c, \epsilon)$ and their effect

- Right shifts: $(q, b) \to (q', c, +1)$, with $b, c \in \Gamma \cup \{¢\}$. When this transition is applied to $¢uqv\$ = ¢uqbv'\$$, the next configuration is $¢ucq'v'\$$. This transition is not applicable to other configurations.
- Left shifts: $(q, b) \to (q', c, -1)$, with $b, c \in \Gamma \cup \{\$\}$. When this transition is applied to $¢uqv\$ = ¢u'aqbv'\$$, the next configuration is $¢u'q'acv'\$$; we assume $u = u'a$ is nonempty (since we do not allow configurations of the form $Q¢\Gamma^*\$$).
- Insertions: $(q, \$) \to (q', c\$, -1)$, with $c \in \Gamma$. When this transition is applied to $¢uq\$$ the next configuration is $¢uq'c\$$.
- Deletions: $(q, c\$) \to (q', c\$, +1)$, with $c \in \Gamma$. When this transition is applied to $¢uqc\$$ the next configuration is $¢uq'\$$.

A machine is called *deterministic* if in every configuration, at most one transition is applicable. A pair of transitions consisting of a left shift followed by a right shift (or a right shift followed by a left shift) is called a *turn*. A one-tape Turing machine is called *sweeping* iff it only makes turns at the endmarkers ¢ and $.

In this paper we always assume that the start state q_0 is a *source* (there is no transitions that leads to q_0), that there is only one accept state, and that this accept state is a *sink* (there is no transition out of this state).

A2. reduction of many tapes to one tape

Theorem. *Any nondeterministic (or deterministic) multi-tape Turing machine with time complexity T(.) and space complexity S(.) is equivalent to a one-tape nondeterministic (resp. deterministic) Turing machine whose time complexity is* $\leq O(T(n) \cdot \max\{n, S(n)\}) \leq O(T(n)^2)$*, and whose space complexity is* $\leq \max\{n, S(n)\} \leq T(n)$*. Moreover, the new one-tape machine is a sweeping machine.*

For a proof see Hopcroft and Ullman (1979, Sections 7.5 and 12.2). The fact that we use finite tapes, and the fact that we want the constructed one-tape machine to be sweeping, leads only to trivial changes in the proof.

A3. universally halting Turing machines and time complexity

We prove a new version of the theorem of Davis (1956), with preservation of time complexity.

Theorem. *Let M be a deterministic Turing machine with time complexity* $\leq T$ *(where T is a total function). Then M is equivalent to a deterministic Turing machine* M' *that always halts after* $\leq O(T(|C|))$ *steps, no matter what configuration* C *the machine* M' *starts in.*

M *is also equivalent to a deterministic* one-tape *Turing machine* M'_1 *that always halts after* $\leq O(T(|C|)^2)$ *steps, no matter what configuration* C *the machine* M'_1 *starts in. Moreover, in a rejecting configuration the tape is empty. And in an accepting computation with input x and output y, every configuration has length at least* $3 + \min\{|x|, |y|\}$.

Proof. Let $M = (Q, \Sigma, \Gamma, ¢, \$, \delta, q_0, q_f)$ be a deterministic (multi-tape) Turing machine; we assume that the start state q_0 is a source, and that there is only one accept state. An

input configuration is of the form $¢q_0w\$$, with all other tapes blank, where w is a word over the input alphabet Σ. Let T be the time-complexity bound of M (when started in input configurations). We will present the proof for the case where M is an acceptor, but the same proof applies to input–output machines.

We add an additional tape to M, called *the history tape* because on this tape we record every transition M makes; on the other tapes the transitions are executed as before. This additional record-keeping multiplies the time-complexity by a constant only, as we shall see. We call the new machine M_h. The history that is recorded has the following purpose: it enables M_h to reverse a computation of M (deterministically, and with just a linear increase in time-complexity), in order to check whether the present configuration is reachable from an input configuration. □

[*Remark*: In an arbitrary configuration of M_h (not necessarily one reached by starting M_h in an input configuration), the content of the history tape can be any sequence of transitions; such a "history" will usually not reflect any well-formed computation. An attempt to reverse such a "history" will usually run into inconsistencies; i.e. the history will provide a transition which could not have been applied before the configuration being considered.]

Based on the machine M_h we will first construct a universally halting Turing machine M'' which halts after $O(T(|C|)^2)$ steps when it starts in any configuration C. After that we will modify M'' to achieve time-complexity $O(T(|C|))$.

The machine M'' has a few additional tapes and executes the following loop:

(1) simulate M_h for a fixed number (say five) of steps, or until a halting configuration of M is reached (whichever comes first);
(2) by using the history of M recorded by M_h, run M (not M_h) in reverse (without altering the recorded history) until the beginning of the history, or until an inconsistency is discovered in the history;
(3) if the last configuration reached during the reverse execution of the history is not an input configuration of M, or if an inconsistency was discovered, M'' rejects and halts;
else (if this last configuration reached is an input configuration of M), M'' starts a new loop (i.e. the head of the history tape is brought back to the right end, the history tape is updated, and the tapes of M'' used to carry out the reverse execution of the recorded history, are erased); however, if in step (1) a halting configuration of M was reached, M'' will halt, thus exiting the loop.

Let us check that the time complexity of M'', starting with any configuration C of M'', is $O(T(|C)^2)$.

The time to execute one loop, beginning with a configuration C_i of M'', is $O(|C_i|)$; indeed, to execute five steps of M_h takes constant time; to execute one transition in the history also takes constant time; and the length of the history contained in configuration C_i is $\leq |C_i|$. If the history corresponds to a well-formed computation of M, starting in an input configuration with input w, then $|C_i| \leq O(T(|w|))$; so in this case one loop takes time $O(T(|w|))$.

If the history in C is inconsistent or does not lead (when executed in reverse) to an input configuration of M, then the very first loop will lead to rejection; and the time to execute a loop is $O(|C|)$, i.e. linear time.

Let us now assume that the history corresponds to a well-formed computation of M, starting with an input configuration with input w. The length of w satisfies

$|w| \leq$ (length of the history) + (length of the current configuration of M).

Indeed, the letters of w that were not erased are still in the current configuration of M, and each letter erased is represented in the history by a letter-deleting transition. Thus $|w| \leq |\mathrm{C}|$. Then M'' will execute the loop at most $\mathrm{O}(T(|w|))(\leq \mathrm{O}(T(|\mathrm{C}|)))$ times. Each loop takes time $\mathrm{O}(T(|w|))$, as we saw for this case. So the total time is $\mathrm{O}(T(|\mathrm{C}|)^2)$.

Let us now modify M'' to obtain a machine M' with time complexity $\mathrm{O}(T(|\mathrm{C}|))$ starting with any configuration C of M'. The machine M' also executes a loop similar to M''; however to speed up the loop, steps (1) and (2) form two *parallel* processes. The parallelism is implemented by letting the two processes execute on two different sets of tapes (including two history tapes).

M' executes the following loop, with (1) and (2) running in parallel:

> Process (1): simulate $\mathrm{M_h}$ until a halting configuration of M is reached, or until process (2) rejects, or until process (2) stops process (1); the history is constantly updated on the right end of the history tape;
> Process (2): using the history of M, run M (not $\mathrm{M_h}$) in reverse until the beginning of the history, or until an inconsistency is discovered in the history; the head on the history tape of process (2) keeps moving left (which corresponds to going back in time);
> if the very last configuration reached (during the reverse execution of the history) is not an input configuration of M, or if an inconsistency was discovered, then M' rejects and halts;
> else (if an input configuration of M is found), M' prepares for starting a new loop: the tapes of process (2) used for carrying out the reverse execution are erased, process (1) is stopped, and the head on the history tape of process (2) is brought back to the right end, while the updated history is copied from the history tape of process (1) to the history tape of process (2);
> now a new loop is started (unless process (1) found a halting configuration of M, in which case M' will halt, thus exiting the loop).

Let us check that the time-complexity of M', starting with any configuration C of M', is $\mathrm{O}(T(|\mathrm{C}|))$.

Again, if the starting configuration C is such that the history record in C is inconsistent or does not lead (when executed in reverse) to an input configuration of M, then the very first loop will lead to rejection; and the time to execute that loop is $\mathrm{O}(|C|)$. So we can assume now that the history corresponds to a well-formed computation of M, starting with an input configuration with input w. The length of w is $\leq |\mathrm{C}|$, for the same reason as before (for M''). In this situation, M' just simulates M in process (1) (while process (2) executes in parallel). Process (1) is stopped while process (2) updates its history tape; process (1) is then idle for as long as the length of its history tape. More precisely, let t_i be the running time of process (1) in the ith run of the loop. Since we deal with a well-formed computation of M on input w, the total running time of process (1) (ignoring its idle times) is $\sum_{1 \leq i \leq m} t_i \leq T(|w|)$, for some number m. The duration of the jth run of the loop is the running time of process (1) (namely t_j), plus the length of the history after j loops (and this is the sum of the past running times of process (1)); thus, the duration of the jth run of the loop is $t_j + \sum_{1 \leq i \leq j} t_i$. Moreover, if loop $j+1$ is not

the last loop, we have $t_{j+1} = \sum_{1 \le i \le j} t_i$ since in its next run, process (1) will not be stopped by process (2) until the whole history has been executed (in reverse). Thus the duration of the jth run of the loop is $t_j + t_{j+1}$, hence the total duration of all the runs of the loop is $\le \sum_{1 \le i \le m}(t_j + t_{j+1}) \le 2T(|w|)$. So the time-complexity of M' is $\mathrm{O}(\max\{|\mathrm{C}|, T(|w|)\}) \le \mathrm{O}(T(|\mathrm{C}|))$.

If we want all tapes (except the output tape) to be empty at the end of any computation, we can achieve this very simply; we may assume that M already erases its tapes at the end of any computation. At the end of every computation (accepting or rejecting), we just let M' enter an erasing state in which it erases the history tape; this takes time $\le |\mathrm{C}_e| \le \mathrm{O}(T(|\mathrm{C}|))$, no matter what configuration C_e the machine is in when the erasing starts.

Finally, to obtain the result about *one-tape* machines of Theorem 1.3, we show the following claim.

Claim. Let M' be a deterministic k-tape Turing machine which is universally halting, and has time-complexity $\le T(|\mathrm{C}|)$ when started in any configuration C. Then the one-tape machine M'_1 obtained by the k-to-one tape conversion (Appendix A2) is also universally halting, and has time-complexity $\le \mathrm{O}(T(|\mathrm{C}_1|)^2)$ when started in any configuration C_1.

Proof of the Claim. M'_1 has two kinds of configurations.

(1) A configuration C_1 of M'_1 may describe a configuration of M', or may be reachable (by transitions of M'_1) from such a configuration. Then the computation of M'_1 that starts with C_1 is a simulation of a computation of M'. Thus, by the universal complexity bound of M', this computation of M'_1 will take time $\mathrm{O}(T(|\mathrm{C}_1|)^2)$.

(2) A configuration C_1 might not be reachable from any configuration of M'_1 that describes a configuration of M'. We claim that in this case, M'_1 will notice that it is not simulating M', after at most two sweeps (i.e. time $\le 2 \cdot |\mathrm{C}_1|$), and halt at that moment.

Recall ((Hopcroft and Ullman, 1979, Section 7.5)) that if a configuration of M'_1 represents a configuration of the k-tape machine M', it has ¢ at the left end, \$ at the right end, and inbetween, the tape is subdivided length-wise into tracks (i.e. the alphabet here consists of k-tuples of letters of M' written as columns). In a track we also use the "blank" symbol # to pad a track, since some tapes of M' may be longer than others. Moreover, on each track there is one position which is "underlined" to indicate the position of the head on this simulated tape. M'_1 simulates one transition of M' by two sweeps.

Any configuration has to use the correct alphabet (corresponding to the simulated tapes), otherwise it is not a configuration of M'_1 at all.

If the padding blanks "#" are not at the right end of each track, M'_1 will recognize this after at most two sweeps (i.e. after $\le 2 \cdot |\mathrm{C}_1|$ steps) and halt (because no transition of M'_1 will be defined).

If all the tracks have the correct padding arrangement, but the underlining representing the head positions are incorrect (i.e. some track has two or more underlinings or has none) then, again M'_1 will recognize this after at most two sweeps.

If the padding and underlining are correct (i.e. the tape of M'_1 represents the tapes of a configuration of M'), the current control state of M'_1 could be inconsistent with the tape content: the state could erroneously remember that some underlined positions have already been seen in the current sweep (although, according to the position and direction of the head of M'_1, they have not), or that some underlined positions have not yet been seen in this sweep (but they have been seen, according to the position and direction of the head of M'_1), or the state remembers the wrong letters at the underlined positions. In

all these cases, M_1' will notice these inconsistencies within two sweeps, and halt in that case.

The last property, that in a rejecting configuration the tape is empty, is easy to obtain: we simply modify the Turing machine so that before halting it erases all tapes (except the output, when the output is valid). And to make sure that in an accepting computation with input x and output y, every configuration is at least as long as $3 + \min\{|x|, |y|\}$, we just have to change the Turing machine so that it never erases the input (except at the end, when the output has been written).

The "$3 + \cdots$" comes from the two endmarkers and the state in the configuration.□

A4. On the complexity of rational relations

Proposition (1) (A version of Eilenberg's cross-section theorem) *If R is a rational subset of $A^* \times B^*$ then there exists a rational partial function $f : A^* \to B^*$ which is a cross section of R (i.e. for every x in the domain of R, $(x, (x)f)$ exists and belongs to R; equivalently,* $\mathrm{dom}R = \mathrm{dom}f$ *and* $f \subseteq R$*).*

(2) Every rational partial function f such that $(1)f = 1$, belongs to F-DTime(linear)*.*

Proof. For (1), which is a fundamental theorem, see Eilenberg (1974, Prop. IX.8.2).

Part (2) follows directly from another important classical theorem (Elgot and Mezei (1965) decomposition theorem; see also Berstel (1979, Theorem IV.5.2)):

A partial function f: $A^* \to B^*$ with $(1)f = 1$ is rational iff f can be written as a composition $(.)f = (.)\lambda\rho$ where $(.)\lambda : A^* \to \sum^*$ is a finite-state sequential length-preserving total function, and $(.)\rho : \sum^* \to B^*$ is a finite-state reverse-sequential partial function (i.e. the function $x \to (x^{\mathrm{rev}})\rho^{\mathrm{rev}}$ is finite-state sequential).

Our proposition follows easily now. Given a rational partial function $(.)f = (.)\lambda\rho$, we construct a deterministic Turing machine which on input x first computes $(x)\lambda$. For this we just simulate a deterministic length-preserving sequential machine; in the process, x is read from left to right and replaced by $(x)\lambda$. Next, the Turing machine moves left while reading $(x)\lambda$, and simulates a deterministic sequential machine which then produces the output $(x)\lambda\rho$.□

Acknowledgements

I thank the referees for their competent and useful reviews; their efforts greatly contributed to this paper.

References

Bauer, G. (1981). Zur Darstellung von Monoiden durch Regelsysteme, Dissertation, Fachbereich Informatik, Universität Kaiserslautern.

Bauer, G. (1985). n-Level rewriting systems. *Theoret. Comput. Sci.*, **40**, 85–99.

Bauer, G., Otto, F. (1984). 'Finite complete rewriting systems and the complexity of the word problem. *Acta Informatica*, **21**, 521–540.

Berstel, J. (1979). *Transductions and Context-Free Languages*. Teubner, Stuttgart.

Boone, W.W., Higman, G. (1974). An algebraic characterization of the solvability of the word problem. *J. Austr. Math. Soc.*, **18**, 41–53.

Birget, J. C. (1996). Two-way automata and length-preserving homomorphisms. *Math. Systems Theory*, **29**, 191–226.

Birget, J. C. (1998) The complexity of the word problem for semigroups and the Higman Embedding Theorem. *Int. J. Algebra Comput.*, to appear.
Bergstra, J. E., Tucker, J. V. (1979). A characterization of computable data types by means of finite equational specification methods, Report IW124/79, Matematisch Centruum, Amsterdam.
Book, R., Otto, F. (1993). *String Rewriting Systems*. Springer, Berlin.
Cohen, D. E. (1989). *Combinatorial Group Theory: A Topological Approach*, LMS **14**, Cambridge University Press, Cambridge.
Craig, W. (1953) On axiomatizability within a system. *J. Symbolic Logic*, **18**, 30–32.
Davis, M. (1956). A note on universal Turing machines. In Shannon, C. E., McCarthy, J., eds, *Automata Studies*, pp. 167–175. Princeton University Press.
Deiss, T. (1993). Conditional semi-Thue systems for presenting monoids. In *Proc. STACS'92* Finkel, A., Jantzen, M., eds, LNCS **577**, pp. 557–565.
Eilenberg, S. (1974). *Automata, Languages and Machines*. Vol. A. Academic Press, New York.
Elgot, C. C., Mezei, J. E. (1965). On relations defined by generalized finite automata. *IBM J. Res. Dev.*, **9**, 47–68.
Epstein, D. B. A., Cannon, J. W., Holt, D. F., Levy, S. V. F., Paterson, M. S., Thurston, W. P. (1992). *Word Processing in Groups*. Jones and Bartlett, Boston.
Evans, T. (1978). An algebra has a solvable word problem if and only if it is embeddable in a finitely generated simple algebra. *Algebra Universalis*, **8**, 197–204.
Gersten, S. M. (1992). Dehn functions and l_1-norms of finite presentations. In Baumslag, G., Miller, C. F., eds, *Algorithms and Classification in Combinatorial Group Theory MSRI Publications*, **23**, Springer, Berlin.
Higgins, P. M. (1992). *Techniques of Semigroup Theory*. Oxford University Press, Oxford.
Higman, G. (1961). Subgroups of finitely presented groups. *Proc. of the Roy. Soc. London, Series A*, **262**, 455–475.
Hodges, W. (1993). *Model Theory*. Cambridge University Press, Cambridge.
Hopcroft, J., Ullman, J. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA.
Jantzen, M. (1988). *Confluent String Rewriting*. Springer, Berlin.
Kashintsev, E. V. (1970). Graphs and the word problems for finitely presented semigroups. *Uch. Zap. Tul. Ped. Inst.* **2**, 290–302 (Russian).
Kobayashi, Y. (1995). A finitely presented monoid which has solvable word problem but no regular complete presentation. *Theoret. Comput. Sci.*, **146**, 321–329.
Lyndon, R., Schupp, P. (1977). *Combinatorial Group Theory*. Springer, Berlin.
Madlener, K., Otto, F. (1985). Pseudo-natural algorithms for the word problem for finitely presented monoids and groups. *J. Symb. Comput.*, LNCS **1**, 383–418.
Madlener, K., Otto, F. (1988). Pseudo-natural algorithms for finitely generated presentations of monoids and groups. *J. Symb. Comput.*, **5**, 339–358.
Madlener, K., Otto, F. (1989). About the descriptive power of certain classes of finite string-rewriting systems. *Theoret. Comput. Sci.*, **67**, 143–172.
Meseguer, J., Goguen, J. A. (1985). Initiality, induction, and computability. In Nivat, M., Reynolds, J., eds, *Algebraic Methods in Semantics*. pp. 459–541. Cambridge University Press, Cambridge.
Murski, V. (1967). Isomorphic embeddability of semigroups with countable sets of defining relations in finitely defined semigroups. *Matematicheskii Zametki*, **1**, 217–224 (English translation, pp. 145–149).
O'Dúnlaing, C. (1983). Infinite regular Thue systems. *Theoret. Comput. Sci.*, **25**, 171–192.
Otto, F. (1989). Restrictions of congruences generated by finite canonical string-rewriting systems. In Dershowitz, N., ed., *Rewriting Techniques and Applications*, 3rd Conf., LNCS **355**, pp. 359–370, Springer, Berlin.
Remmers, J. H. (1980). On the geometry of semigroup presentations. *Advances in Math.*, **36**, 283–296.
Sims, C. C. (1994). *Computation with Finitely Presented Groups*. Cambridge University Press, Cambridge.
Squier, C. C. (1987). Word problems and a homological finiteness condition for monoids. *J. Pure Appl. Algebra*, **49**, 201–217.
Squier, C. C., Otto, F. (1987). The word problem for finitely presented monoids and finite canonical rewriting systems. In Lescanne, P., ed., *Rewriting Techniques and Applications*, 2nd Conf. LNCS **256**, Berlin, Springer.
van Leeuwen, J. (ed.) (1990). *Handbook of Theoretical Computer Science*. Vols A and B. MIT Press/Elsevier.

Originally received 20 September 1995
Accepted 3 October 1997